Système de Mesure Mobile Adaptif Qualifié

Florent Bourgeois

École doctorale pluridisciplinaire Jean-Henri Lambert, nº 494

THÈSE

pour obtenir le grade de docteur délivré par

l'Université de Haute Alsace

Spécialité doctorale : "Informatique"

présentée et soutenue publiquement par

Florent BOURGEOIS

le 21 mars 2018

Système de Mesure Mobile Adaptif Qualifié

Directeur de thèse : **Jean-Marc PERRONNE**
Co-encadrant de thèse : **Philippe STUDER**

Jury

M. Fabrice Bouquet,	
Professeur; Université de Bourgogne Franche comté	Rapporteur
M. Amir Hajjam El Hassani,	
Maitre de conférence HDR; Université de Belfort-Montbéliard	Rapporteur
M. Jean-Pierre Bourey,	
Professeur; Université des Sciences et Technologies de Lille	Examinateur
M. Philippe Studer,	
Maitre de conférence; Universitée de Haute Alsace	Examinateur
M. Jean-Marc Perronne,	
Professeur; Universitée de Haute Alsace	Directeur de thèse
M. Olivier Grzelak,	
Chef de Projet; Actimage GmbH	Invité

Laboratoire Modélisation Intelligence Processus et Systèmes (MIPS) – EA2332
12 rue des Frères Lumière, F-68093 Mulhouse (France)

Remerciements

Avec ce manuscrit se clôt un travail de cinq années. Ce sont cinq années d'expériences scientifiques, professionnelles, d'enseignements et personnelles. J'y ai découvert les métiers de chercheur, d'ingénieur, d'enseignant, d'oncle et de conjoint. J'aime maintenant à dire que la thèse est une aventure de la vie professionnelle ; entrecoupée de hauts et de bas intenses. L'enfer s'ouvre sous vos pieds pendant les phases de rédactions ou lorsqu'un verrou technique est abordé. Mais le paradis ouvre ses portes lors de l'acceptation d'un papier ou quand la solution au verrou est trouvée. Plusieurs personnes m'ont accompagné durant cette aventure et je tiens à les remercier.

Je remercie chaleureusement toutes les personnes qui m'ont aidé pendant l'élaboration de ma thèse et notamment mes encadrants. Jean-Marc Perronne, pour ses conseils et sa patience concernant mes travaux de recherche, mais aussi pour la confiance qu'il a placé en moi en me permettant de rejoindre l'équipe d'enseignement de l'ENSISA. Philippe Studer, pour ses réponses concises toujours judicieuses, sa disponibilité et son accompagnement lors de chaque étape de ma thèse. Bernard Thirion, qui m'a donné envie de réaliser une thèse pour devenir enseignant, pour avoir accepté de m'assister tout au long de mes travaux, pour le temps qu'il y a passé et pour nos discussions toujours passionnées. Merci aux doctorants et stagiaires (Mariem, Houda, Trung, Heng, Baptiste, Paul, Oleg) avec qui j'ai découvert de nouvelles cultures, et survécu à de longues journées, aux températures parfois alarmantes, interrompues par une fermeture venant trop tôt les soirs de deadline. J'ai également une pensée pour tous les collègues de l'ENSISA qui ont su me guider, m'encourager et me vider la tête.

Je remercie également Actimage GmbH qui a accepté de me donner la chance de réaliser ces travaux en me guidant vers ce projet. J'ai une pensée particulière pour : Olivier, Alexandra et Ashley qui ont pris le temps de m'orienter de répondre à mes questions et calmer mes incertitudes ; Félicien, mon mentor sur les technologies de la pomme vers qui je me tournerais, encore aujourd'hui pour résoudre un conflit git ; Pierre Arlaud qui a rendu les journées de travail bien plus agréables à l'aide de Bernie Noël, de musiques de tout genre et d'Ecco ; chacun des autres membres de l'équipe qui ont apporté leurs pierres à l'édifice.

À tous mes amis, merci d'avoir accepté ces années d'indisponibilité. Malgré tout, j'ai vécu de sacrées aventures avec vous. Thibaut, vous m'avez fait découvrir la magie de Nice et le rôle de témoin, une des meilleures signatures de ma vie. Bruno, tu n'imagines pas le nombre de repas "taktaktak" que j'ai mangé depuis Besançon ; sans compter nos multiples débats sur

l'enseignement. Les Haskulls, des WE à Gray, des vacances dans le sud, des scorpions, des Shaka Ponk et des Nightwish. Les GEII ; paradoxalement à la faible distance ; vous êtes ceux qui m'ont le moins vu. Mais je me soigne et nous avons partagé pas mal de grandes nouvelles en septembre. Brendan, que dire de mon ami et coéquipier de l'IUT, de la JMU et de l'ENSISA ? Tu as sûrement été, malgré toi, la personne à m'avoir le plus dirigé vers ce succès. Antoine et Antoine, merci l'un d'avoir été le meilleur coloc', l'autre mon partenaire de soirées, de musique et de re-modélisation du monde.

Je me tourne maintenant du côté de ma famille. À toute la famille Zuchowicz, mon second exploit de ces 5 années a été de savoir écrire votre nom. Avec vous tous, j'ai découvert ce qu'étaient les rôles de beau-frère, d'oncle et de gendre. Je ne sais pas si je les joue bien. En tous cas, c'est un plaisir de tous vous compter dans ma famille.

Un grand merci à mes oncles, tantes et grands-parents, pour toutes les leçons de vie qu'ils ont pu m'enseigner au cours des nombreux repas, jeux, balades, week-ends et vacances que j'ai pu passer avec chacun d'entre vous. Ainsi qu'à mes cousins et cousines (et je compte les pièces rapportées, moi) avec lesquels, on commence à prendre le relais. On a déjà eu un paquet d'aventures ensemble et j'espère bien d'autres à venir.

Maman, papa, vous m'avez montré la voie pour que je grandisse. Vous avez su m'accompagner lorsque j'ai commencé à déployer mes ailes et maintenant vous m'accompagnez fièrement vers l'horizon. Merci de m'avoir offert toutes les opportunités qui s'offraient à moi. Je vous en serai toujours reconnaissant. N'oubliez pas que vous aussi vous pouvez compter sur moi maintenant.

Paul, qu'est-ce qu'on a pu se chamailler plus jeunes. Mais on est et on reste le meilleur duo de 101% qui n'a jamais existé ! En plus, tu ne m'as jamais demandé si j'avais bientôt terminé ma thèse… Le mot bro n'aura jamais eu autant de sens que là. Maintenant que j'ai terminé ma thèse, ça te tente un petit Borderland ?

Enfin, jamais cette montagne n'aurait été franchie sans l'aide inconditionnelle et les nombreux sacrifices de ma petite fiancée. Je vais enfin devenir *ton* Docteur. Et surtout… papa.

À tous ceux que j'ai cités, c'est grâce à vous tous que j'ai su devenir la personne que je suis aujourd'hui et que j'ai réussi à aller au bout de ce projet complètement fou qu'est la thèse.

Je dédie ce manuscrit à mon grand-père. Il m'aurait certainement dit avec justesse, le regard plein de fierté, cette phrase qui m'a toujours poussé plus loin : *Tu as vu là, tu aurais pu faire mieux.*

Résumé

Les dispositifs matériels mobiles proposent des capacités de mesure à l'aide de capteurs soit embarqués, soit connectés. Ils ont vocation à être de plus en plus utilisés dans des processus de prises de mesures. Ils présentent un caractère critique dans le sens où ces informations doivent être fiables, car potentiellement utilisées dans un contexte exigeant.

Malgré une grande demande, peu d'applications proposent d'assister les utilisateurs lors de relevés exploitant ces capacités. Idéalement, ces applications devraient proposer des méthodes de visualisation, de calcul, des procédures de mesure et des fonctions de communications permettant la prise en charge de capteurs connectés ou encore la génération de rapports. La rareté de ces applications se justifie par les connaissances nécessaires pour permettre la définition de procédures de mesure correctes. Ces éléments sont apportés par la métrologie et la théorie de la mesure et sont rarement présents dans les équipes de développement logiciel. De plus, chaque utilisateur effectue des activités de mesure spécifiques au domaine de son champ d'activités, ce qui implique le développement d'applications spécifiques de qualité pouvant être certifiées par des experts.

Ce postulat apporte la question de recherche à laquelle les travaux présentés répondent : *Comment proposer une approche pour la conception d'applications adaptées à des procédures de mesures spécifiques. Les procédures de mesure pouvant être configurées par un utilisateur final.*

La réponse développée est une "plateforme" de conception d'applications d'assistance à la mesure. Elle permet d'assurer la conformité des procédures de mesures sans l'intervention d'expert de la métrologie. Pour cela elle est construite en utilisant des concepts issus de la métrologie, de l'Ingénierie Dirigée par les Modèles et de la logique du premier ordre.

Une étude du domaine de la métrologie permet de mettre en évidence la nécessité d'une expertise des procédures de mesure impliquées dans les applications. Cette expertise comprend des termes et des règles assurant l'intégrité et la cohérence d'une procédure de mesure. Un modèle conceptuel du domaine de la métrologie est proposé. Ce modèle conceptuel est ensuite intégré au processus de développement d'une application. Cette intégration se fait par un encodage de ce modèle conceptuel sous la forme d'un schéma des connaissances de la métrologie en logique du premier ordre. Il permet, la vérification du respect des contraintes inhérentes à la métrologie dans une procédure de mesure. Cette vérification est réalisée en confrontant les procédures de mesures au schéma sous forme de requêtes. Ces requêtes sont décrites à l'aide d'un langage proposé par le schéma.

Les applications d'assistance à la mesure nécessitent d'exposer à l'utilisateur un processus de mesure impliquant relevés et affichages de mesures étape par étape. Cela implique de pouvoir décrire un processus de mesure et d'en définir les interfaces et le schéma d'évolution. Pour cela, un éditeur d'application est proposé. Cet éditeur propose un langage spécifique dédié à la description d'applications d'assistance à la mesure. Ce langage est construit à partir des concepts, formalismes et outils proposés par l'environnement de méta-modélisation Diagrammatic Predicate Framework (DPF). Le langage comporte des contraintes syntaxiques prévenant les erreurs de construction au niveau logiciel tout en réduisant l'écart sémantique entre l'architecte logiciel l'utilisant et un potentiel expert de la métrologie.

Enfin, les mobiles doivent pouvoir exécuter un comportement conforme à celui décrit avec l'éditeur. Pour cela, un langage de modélisation d'implantations est proposé. Il permet de décrire les procédures de mesures sous la forme de séquences d'activités impliquant des relevés, des calculs et des présentations de valeurs. Les grandeurs sont abstraites par des valeurs numériques, simplifiant leurs manipulations et l'utilisation des différents capteurs du marché. Le modèle d'implantation est constitué d'agents. Une application mobile est également proposée. Elle est construite autour d'un framework d'agents, d'un constructeur de réseaux d'agents et d'un moteur d'exécution. Elle prend en entrée un modèle d'implantation, construit le réseau d'agents correspondants et ainsi propose un comportement qui répondra aux besoins de l'utilisateur. Ce qui permet de répondre aux besoins de tout utilisateur, du moment qu'il ait accès au modèle d'implantation, sans pour autant nécessiter de télécharger de nombreuses applications.

Les trois outils présentés, l'éditeur de modèles, le schéma de connaissance de la métrologie et l'application mobile manipulent des modèles différents. Chaque modèle représente un point de vue spécifique sur l'application. Les concepts de (méta)modélisation et de transformation de modèles proposés par l'Ingénierie Dirigée par les Modèles, et plus particulièrement DPF, rendent possible l'interopérabilité automatisée des trois outils. La métamodélisation permet l'emploi du modèle conceptuel de la métrologie comme noyau architectural commun aux trois outils. Les transformations de modèles permettent de transposer les concepts d'un modèle vers ceux d'un autre, afin de naviguer entre les différentes vues de l'application employées par les outils. Cela permet la création d'une "plateforme" de conception d'application simplifiant le travail de développement tout en assurant la qualité des applications générée.

La valorisation de ces concepts au sein d'un projet industriel est illustrée par le projet Actinote 4.0 d'Actimage GmbH, le partenaire industriel initiateur de ces travaux.

Abstract

Mobile devices offer measuring capabilities using embedded or connected sensors. They are more and more used in measuring processes. They are critical because the performed measurements must be reliable because possibly used in rigorous context.

Despite a real demand, there are relatively few applications assisting users with their measuring processes that use those sensors. Such assistant should propose methods to visualise and to compute measuring procedures while using communication functions to handle connected sensors or to generate reports. Such rarity of applications arises because of the knowledges required to define correct measuring procedures. Those knowledges are brought by metrology and measurement theory and are rarely found in software development teams. Moreover, every user has specific measuring activities depending on his field of work. That implies many quality applications developments which could request expert certification.

This premises bring the research question the presented works answer : *What approach enables the conception of applications suitable to specific measurement procedures considering that the measurement procedures could be configured by the final user.*

The presented works propose a platform for the development of measuring assistant applications. The platform ensure the conformity of measuring processes without involving metrology experts. It is build upon metrology, model driven engineering and first order logic concepts.

A study of metrology enables to show the need of applications measuring process expert evaluation. This evaluation encompasses termes and rules that ensure the process integrity and coherence. A conceptual model of the metrology domain is proposed. That model is then employed in the development process of applications. It is encoded into a first order logic knowledge scheme of the metrology concepts. That scheme enables to verify that metrology constraints holds in a given measuring process. The verification is performed by confronting measuring processes to the knowledge scheme in the form of requests. Those requests are described with a request language proposed by the scheme.

Measuring assistant applications require to propose to the user a measuring process that sequences measuring activities. This implies to describe a measuring process, and also to define interactive interfaces and sequencing mecanism. An application editor is proposed. That editor uses a domain specific language dedicated to the description of measuring assistant applications. The language est build upon concepts, formalisms and tools proposed by the metamodeling environment : Diagrammatic Predicat

Framework (DPF). The language encompasses syntactical constraints that prevent construction errors on the software level while reducing the semantical gap between the software architect using it and a potential metrology expert.

Then, mobile platforms need to execute a behaviour conforming to the editor described one. An implementation modelling language is proposed. This language enables to describe measuring procedures as sequences of activities. Activities imply to measure, compute and present values. Quantities are all abstracted by numerical values. This eases their computation and the use of sensors. The implementation model is made up of software agents. A mobile application is also proposed. The application is build upon a framework of agents, an agent network composer and a runtime system. The application is able to consider an implementation model and to build the corresponding agent network in order to propose a behaviour matching the end users needs. This enables to answer to any user needs, considering he can access to the implementation model, without requiring to download several applications.

The three tools presented : the application editor, the metrology knowledge scheme and the mobile application handle different models. Every model represent a specific point of view on the application. Concepts of (meta)modelling and model transformations proposed by Model Driven Engineering, especially those of DPF, enables automated interoperability of those three tools. Metamodelling enable to use the metrology conceptual model as a common architectural core. Model transformations enable to transpose concepts of a model toward those of another, in order to navigate between the different point of view of the application employed by each tool. This enables the creation of a platform for the development of applications dedicated to assist software development while ensuring the quality of created applications.

Those concepts have been promoted through an industrial project : Actinote 4.0 from Actimage GmbH, these works industrial partner.

Table des matières

Liste des figures

Liste des tableaux

Liste des codes

Chapitre 1

Introduction

« *Il n'est que de tout mesurer pour perdre le sens de la mesure.* »

Robert Sabatier

Sommaire

1.1 Motivations et enjeux

1.1.1 Contexte

La collecte de données est une problématique présente dans la plupart des domaines (métier du bâtiment, cartographie, assistance au diagnostic, design intérieur, ...). L'acquisition peut être faite en une fois (métré) ou être périodique, afin d'acquérir une évolution, une tendance. Le contexte socio-économique (services, disponibilité de réponses en vis à vis) et la démocratisation des interventions à domicile (soins, devis) font que la mobilité est de plus en plus demandée. Pour répondre à cette demande, des systèmes légers, portables, minimisant l'équipement nécessaire et capable de collecter les données de manière suffisamment efficace sont demandés.

Le marché répond à cette situation en proposant des systèmes portables (smartphones, tablettes) construits autour de dispositifs dédiés à la mesure (boussole, accéléromètre, ...). La collecte de mesures est rendue possible par un nombre croissant d'applications. Ces applications exploitent globalement de manière satisfaisante les capacités offertes par les systèmes. En complément, afin d'étendre les capacités de ces systèmes, des dispositifs matériels communicants spécialisés apparaissent.

Cependant, les solutions existantes sont souvent inadéquates et insuffisantes. Elles sont souvent limitées en ne proposant de réaliser que des mesures très spécifiques à un domaine (mesurer un angle, un volume, ...). Un utilisateur doit donc continuellement passer d'une application à l'autre comme s'il changeait d'instrument de mesure. Il est dans l'obligation de faire un traitement à postériori pour calculer le résultat désiré. D'autres solutions sont trop spécifiques car proposant une réponse à un processus de mesure complexe ; elles demandent de suivre un protocole rigide afin d'accomplir une tâche spécifique.

Cet état de fait s'explique par l'écart existant entre l'expertise requise pour la définition d'une procédure de mesure et sa vulgarisation induite par l'actuelle généralisation de l'accès à des capteurs complexes pour tout utilisateur. Ceci s'exprime au travers de deux points principaux :

— Obtenir un résultat de mesure correct implique l'utilisation d'une procédure de mesure spécifiquement adaptée. Cependant, dans la pensée commune, ce résultat est une simple valeur numérique manipulable par toutes opérations algébriques,

— La documentation d'un instrument de mesure est complexe. Elle précise ses limites d'utilisation et les protocoles d'étalonnages pour en garantir l'utilisation. En revanche, la plupart de ces instruments pré-

sentent à l'utilisateur une interface simple permettant de relever des valeurs, souvent numériques.

Les résultats de mesurage (action de mesurer) interviennent dans plusieurs domaines (production, santé, économie, ...). Ce qui implique que cette vulgarisation du mesurage peut être un danger. En effet, un résultat erroné peut impliquer des conséquences [118] financières ou même humaines. C'est pourquoi les travaux présentés visent à démontrer l'importance de la métrologie dans des contextes de développement logiciels et proposent une solution permettant de vérifier la cohérence des procédures de mesures. Cette solution est développée dans un contexte de génération d'applications d'assistance à la mesure configurées pour répondre spécifiquement aux besoins de chaque utilisateur. Ainsi, l'application vise à offrir à l'utilisateur professionnel, comme au particulier, un ensemble d'outils lui permettant d'atteindre ses objectifs tout en prenant en compte le contexte de la mesure.

Le travail de thèse s'inscrit dans le contexte industriel de la société Actimage GmbH [1]. Actimage GmbH est un éditeur de logiciels proposant des solutions de prise d'information en situation de mobilité afin que les données collectées soient "poussées" dans les systèmes d'informations des clients. La préoccupation étant d'établir en fonction des domaines d'activité, des diagnostics, les documents préparant une production, une commande, ...

Les travaux ont dû prendre en compte en plus de la dimension recherche les aspects de faisabilité et de mise en oeuvre afin d'aboutir à des méthodes et solutions valorisables par la société. Cet aspect concret implique de prendre en compte les limitations liées aux systèmes, existants ou à venir, le monde des plateformes mobiles évoluant rapidement.

1.1.2 Analyse des besoins

Le contexte spécifique des travaux est décrit au travers d'une description des besoins de chacune des quatre parties prenantes impliquées : l'organisme de développement, l'utilisateur final, le spécialiste de la métrologie et l'architecte logiciel.

L'organisme de développement

Actimage GmbH cherche à produire un ensemble d'applications dédiées à l'assistance d'opérateurs dans des contextes de mobilité. Elles peuvent servir, par exemple, à estimer le coût d'installation de panneaux solaires,

1. http://www.actimage.de

ou encore à établir des diagnostics sur l'état d'une installation industrielle. Après étude de leurs besoins, ils nécessitent que l'application proposée à un utilisateur instrumentalise une procédure de mesure qui :

— produit les grandeurs requises par l'utilisateur,
— prend en charge les instruments et les méthodes spécifiques de l'utilisateur,
— propose de relever plus que de simples mesures traditionnelles (informations client, style d'intérieur, couleurs, ...).

Ces applications présentent des caractéristiques similaires (e.g. construction autour d'une procédure de mesure, hautement configurable, conforme à la métrologie). Actimage GmbH a besoin d'un processus de conception et de création d'applications qui soit rapide, assurant ces qualités et caractéristiques.

L'utilisateur final

L'utilisateur final est un habitué de la prise de mesure. Il emploie des procédures de mesures quotidiennement lors de ses activités. Il possède un ensemble d'instruments spécialisés, utilise un vocabulaire propre à son secteur d'activité et son expérience de la mesure vient du terrain.

L'utilisateur est celui qui exprimera les besoins spécifiques de l'application. C'est avec ses mots qu'il énoncera une description de la ou les procédure(s) de mesure qu'il espère voir assistée(s) par un système logiciel mobile.

La description formulée contient des informations sur le matériel à disposition. Cela comprend le système mobile, les instruments de mesure et les systèmes de visualisation que l'utilisateur envisage d'utiliser. La description présente également des prescriptions concernant les grandeurs et unités impliquées aux entrées et sorties de la procédure de mesure. Enfin, il est possible que l'utilisateur demande de respecter des contraintes d'incertitude de mesure sur les résultats obtenus.

L'aspect mobile prend tout son sens pour une telle application. En effet, l'utilisateur pourrait facilement l'avoir à portée de main comme un outil supplémentaire, ou même à la place d'autres outils.

Pour être valide aux yeux de l'utilisateur final, l'application doit pouvoir :

— Produire les résultats requis,
— employer des méthodes et des instruments spécifiques maîtrisés par l'utilisateur,
— proposer un résultat conforme aux contraintes exprimées,
— être adaptée à la plateforme mobile ciblée.

L'expert de la métrologie

Les différents concepts de la métrologie impliqués dans ce projet sont : la description de procédures de mesure, leurs vérifications, l'évaluation des incertitudes de mesures et également la prise en charge de relevés non physiques. En effet, les grandeurs non physiques sont indissociables des domaines des utilisateurs (e.g. type de tuiles sur un toit).

Ni l'utilisateur final ni l'architecte logiciel ne sont experts en métrologie. Il faut donc proposer une approche accessible par des néophytes capables de garantir la qualité de l'application générée. C'est donc l'approche qui devra prendre en charge le rôle de l'expert de la métrologie. Cela implique des capacités de validation d'une procédure de mesure en fonction des différentes entrées, instruments, opérations et sorties impliquées, quel que soit le domaine spécifique adressé.

L'architecte logiciel

L'architecte logiciel spécifie l'application à partir de la description des besoins de l'utilisateur. Ses connaissances en métrologie sont considérées faibles et il communique avec le vocabulaire du domaine logiciel. Il y a donc un écart sémantique entre les termes qu'il maîtrise et ceux composant la description. De plus, la description spécifie les activités de mesure ainsi que leurs ordonnancements.

L'architecte doit être capable de comprendre les différents éléments constituant la description afin d'ensuite les inclure dans une spécification de l'application. Pour cela, il faut que l'approche :

— réduise l'écart sémantique entre les vocabulaires de l'utilisateur et de l'architecte,

— propose de composer des séquences d'activités.

La figure 1.1 illustre les différentes parties prenantes et les produits impliqués dans le développement d'une application dans le contexte des travaux réalisés.

1.1.3 Questions de recherche

Le postulat décrit par l'analyse des besoins permet de définir le cadre dans lequel nos travaux ont été réalisés. La problématique fondamentale à laquelle ces travaux tendent à trouver une réponse est la suivante :

Comment proposer une approche pour la conception d'applications adaptées à des procédures de mesures spécifiques ? Les procédures de mesure pouvant être configurées par un utilisateur final.

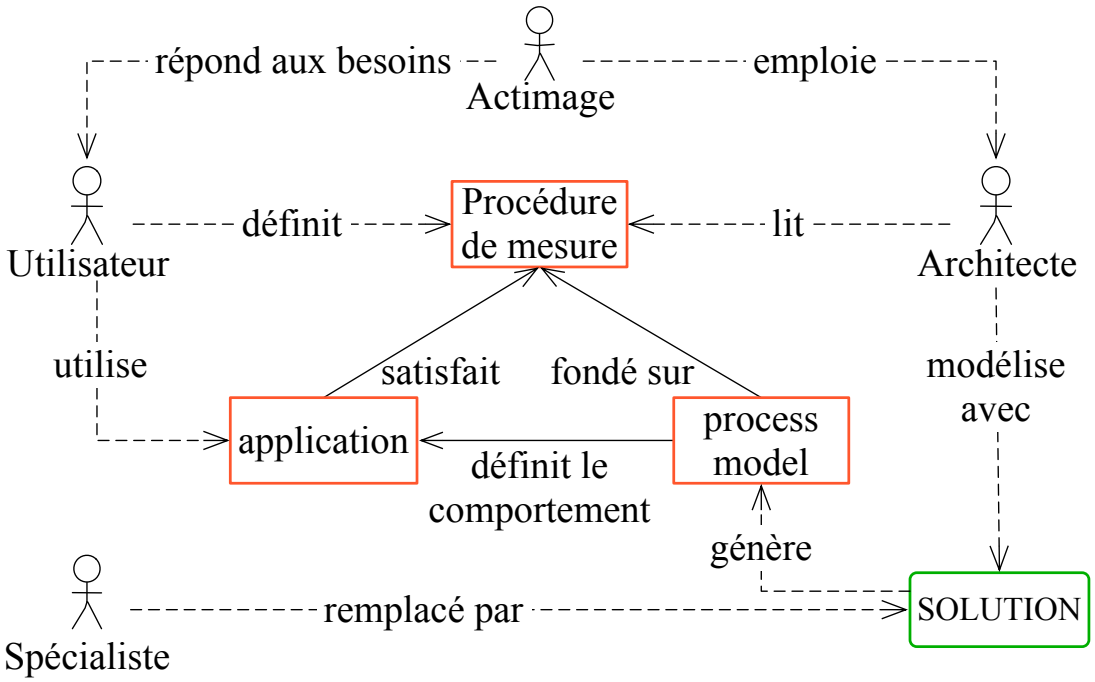

Figure 1.1 – Acteurs et produits interagissant dans la solution proposée

Celle-ci peut être décomposée en trois problématiques :

— Qu'est-ce qu'une procédure de mesure et comment en garantir la cohérence ?

— Comment intégrer des procédures de mesure dans un logiciel ?

— Quelle approche permettrait de faciliter la conception de logiciel dans ce domaine ?

1.2 Contribution

Nos travaux cherchent à répondre à ces questions. Pour cela, la science de la mesure, appelée métrologie, est étudiée. Les fondements de cette science, ces standards, le vocabulaire ainsi que les règles qu'elle décrit sont abordés. De plus, la théorie représentationnelle du mesurage (representational theory of measurement : RTM [70; 119]) est présentée. C'est une approche, plus ouverte que les standards de la métrologie, qui permet d'ajouter à la métrologie l'utilisation de grandeurs non physiques. Ces présentations aboutissent à la proposition d'un modèle conceptuel pour les mesurages.

Le modèle conceptuel est ensuite utilisé comme base pour inférer un schéma des connaissances du domaine [32] dans l'espace de la logique du

premier ordre [77]. En plus d'encoder les éléments de la métrologie, le schéma offre des capacités de raisonnement. Ce raisonnement permet de vérifier la conformité d'une procédure de mesure ou d'une opération de conversion par rapport aux règles inférées. Cette vérification permet d'assurer la cohérence d'une procédure de mesure impliquant différents capteurs, types de grandeurs, unités résultats et manipulations de résultats de mesures. La vérification est réalisée en présentant au schéma une procédure de mesure sous la forme d'une requête qu'il réduira à l'aide d'un algorithme d'unification. Ces requêtes ne peuvent être valides que si elles sont décrites en employant les termes proposés par le schéma de connaissance. Ce qui implique qu'un langage de requêtes de validation de procédures de mesure est induit par le schéma. Ce langage est nommé le Measurement Knowledge Query Language (MKQL).

Un langage dédié, ou *Domain Specific Language* (DSL) [45] pour la description d'applications logicielles fondées sur des procédures de mesure est proposé. Il est nommé le langage de modélisation d'application d'assistance à la mesure, Measure Application Modelling Language (MAML). L'utilisation d'un DSL permet de simplifier la syntaxe disponible tout en augmentant la sémantique [56] associée au contexte. MAML propose des termes provenant des domaines des processus, des interfaces utilisateur et les mêle à des termes de la métrologie. Il permet ainsi de réduire l'écart sémantique entre un architecte logiciel et un professionnel réalisant des mesurages quotidiennement. Les modèles qui en sont issus décrivent une procédure de mesure sous la forme de séquences d'activités de mesurages, de manipulation de grandeurs et de présentation de résultats.

D'un point de vue logiciel, une mesure est largement représentée sous la forme d'un simple nombre. Cette représentation fait écho à un manque d'uniformisation de la gestion des grandeurs dans les systèmes d'information (capteurs, logiciels) [42]. Une approche permettant d'assurer la prise en charge de grandeurs dans un logiciel est proposée. Elle propose d'uniformiser les valeurs obtenues et ainsi de simplifier leurs manipulations. Un second DSL, cette fois dédié au côté fonctionnel de la procédure de mesure est proposé. Ce langage est nommé le langage pour la description de configuration d'application de mesure, ou Executable Measurement Application Model Language (EMAML). L'EMAML permet de décrire un modèle à un niveau d'abstraction plus bas que le précédent. Les éléments propres à la métrologie ne sont pas présents, cependant il propose une syntaxe dont la sémantique est exécutable par un système logiciel. Les modèles qui en sont issus sont donc constitués de séquences de requêtes de données provenant de divers systèmes (capteurs, bases de données, interfaces utilisateur, ...), d'uniformisation des données, d'opérations algébriques sur ces données et

de présentation des données.

Une approche permettant l'aide la création d'application d'assistance à la mesure doit proposer les capacités de chacun des trois langages proposés. En effet, les trois activités majeures d'un cycle de développement sont : la *spécification*, la *vérification* et l'*implantation* [12]. Ces travaux proposent d'intégrer ces langages sous la forme de trois outils, l'*éditeur*, le *vérificateur* et l'application mobile, chacun prenant en charge l'une de ces activités. Seulement, les trois langages proposent un point de vue différent sur ce qu'est une procédure de mesure.

L'intégration de ces outils au sein d'une plateforme est réalisée par les méthodes proposées par l'*Ingénierie Dirigée par les Modèles* (IDM), ou Model Driven Engineering. L'IDM [17; 115] est un paradigme de développement qui emploie les modèles comme entités de premier ordre. Elle propose également le concept de transformations de modèles permettant d'adapter un modèle en un nouveau. L'intérêt étant de proposer un modèle focalisé sur l'activité à réaliser par chaque outil de la plateforme. Ces transformations permettent donc de traduire le modèle issu de la spécification en une requête vérifiable ou encore en un modèle exécutable. Ces deux transformations sont respectivement nommées Model to Verification Query (M2VQ) et Model to Application (M2APP).

La plateforme est nommée *MEASURE* et sa structure présentée dans la figure 1.2.

L'architecte logiciel utilise la plateforme MEASURE pour spécifier l'application qui remplira les besoins définis par l'utilisateur final.

— l'*éditeur* est utilisé par l'architecte logiciel pour décrire le *modèle de spécification* d'une application. Le langage MAML, qu'il emploie, propose des termes compris dans le vocabulaire de la métrologie, ce qui réduit l'écart sémantique entre l'utilisateur et l'architecte logiciel.

— Le *vérificateur* est employé pour identifier les erreurs relatives à la métrologie dans une procédure de mesure. Il utilise pour cela les capacités de raisonnement du schéma des connaissances de la métrologie.

— l'*application mobile* est le système logiciel mis à disposition de l'utilisateur final. Ce système utilise un modèle exécutable pour proposer un comportement sur mesure répondant aux besoins spécifiques de l'utilisateur final.

— l'*extracteur de requêtes* nommé M2VQ permet de transposer un modèle de spécification en requête de vérification. Cela rend possible la vérification de la procédure de mesure au travers d'un raisonnement autour de la sémantique de la métrologie.

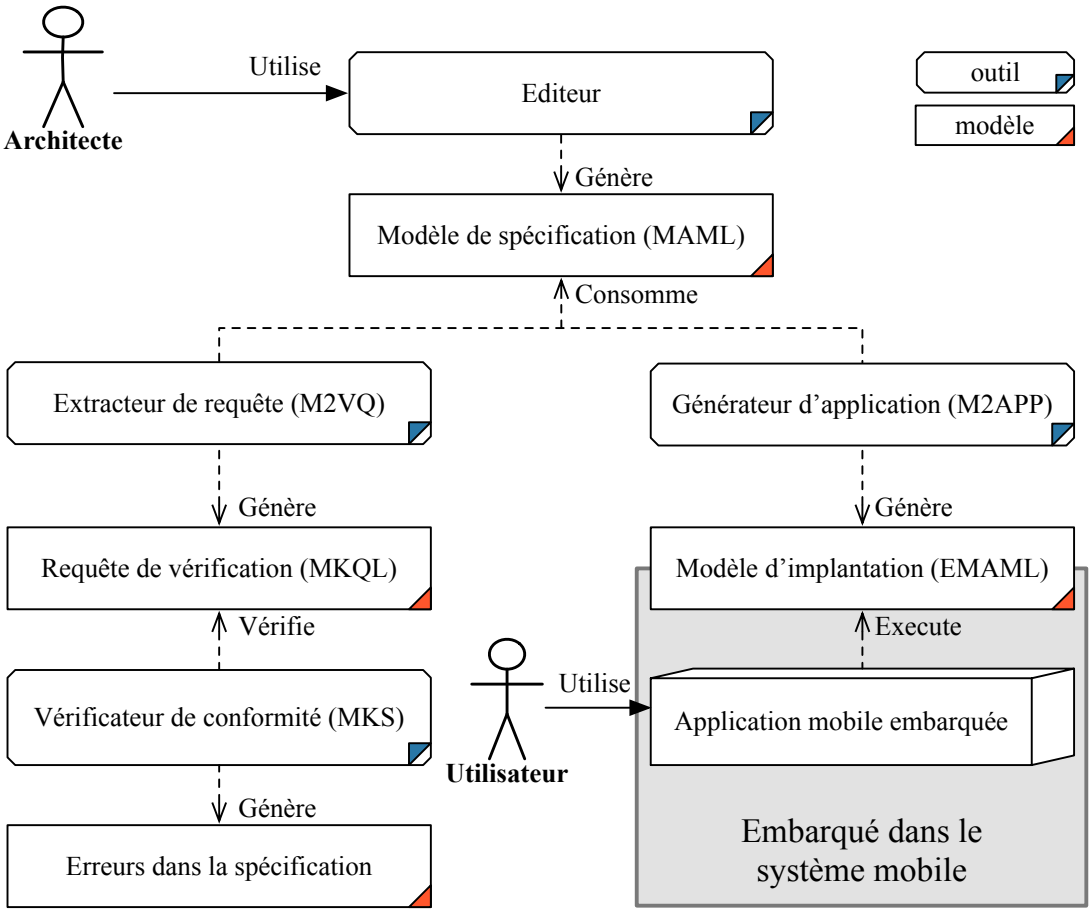

Figure 1.2 – Outils et modèles interagissant dans la plateforme MEASURE

— le *générateur d'application* nommé M2APP permet de générer un modèle exécutable de l'application à partir d'un modèle de spécification. Cela permet d'ajouter une sémantique exécutive à la spécification tout en retirant la sémantique liée à la métrologie, inutile dans le contexte fonctionnel.

1.3 Plan

Le manuscrit de thèse est organisé en sept chapitres :

Le **chapitre 2** présente la métrologie en abordant les termes et les règles employés dans ce domaine. Il a pour but de démontrer l'apport indispensable que peut proposer une expertise en métrologie lors de la création de systèmes logiciels. Dans un premier temps il présente l'intérêt des efforts d'uniformisation qui ont été réalisés depuis plusieurs années pour simplifier la prise de mesure. Cette simplification a eu un effet pervers qui sera également illustré. Ensuite, il présente le contexte socio-économique actuel qui propose un nombre croissant de systèmes de mesures. Puis, le chapitre décrit les termes composant le langage du métrologue et les structures imaginées pour simplifier la représentation d'une procédure de mesure. À ces termes et structures s'ajoutent ensuite les contraintes et règles permettant d'assurer que le résultat d'un mesurage est cohérent. Enfin, le chapitre se conclut en proposant une description des différentes solutions logicielles permettant d'augmenter la confiance que l'on peut avoir dans une procédure de mesure.

Le **chapitre 3** présente les différentes approches du génie logiciel qui sont employées pour réaliser ces travaux. Il décrit les différentes approches permettant l'inférence de connaissances dans des systèmes logiciels. Ensuite, il présente les approches d'implantations sous forme d'entités logiques des trois paradigmes logiciels : objet, composant et agent. Et enfin, il aborde l'Ingénierie Dirigée par les Modèles, une approche de description de logiciel utilisant les modèles comme entités de description principales.

Le **chapitre 4** présente le Metrology Knowledge Schema (MKS), un schéma des connaissances de la métrologie. Ce schéma encode les différentes règles et contraintes composant la science de la mesure en logique du premier ordre. Il permet de raisonner dans le domaine de la métrologie et donc de vérifier la cohérence de procédures de mesure. Pour commencer, il propose un modèle conceptuel pour la représentation de procédures de mesures fondé sur les standards du domaine. Ensuite, il présente une inférence en logique du premier ordre des termes et relations de ce

modèle. Après quoi, il montre l'ajout des contraintes et règles inhérentes à la métrologie. Celles-ci comprennent les règles et contraintes d'analyse pour valider des procédures de mesure et celles assurant des conversions entre références d'un même type de grandeur. Enfin, il décrit comment la combinaison de ces règles permet au schéma de connaissances de vérifier des procédures de mesure faisant interagir des grandeurs aux unités hétérogènes. Cette vérification est réalisée en décrivant des procédures de mesures sous la forme de requêtes conformes au MKQL, le langage de requêtes induit par le schéma de connaissances.

Le **chapitre 5** présente la plateforme MEASURE. Cette plateforme propose un cycle assisté de développement d'application d'assistance à la mesure qualifiées. Le cycle présente trois activités : la *spécification*, la *vérification* et l'*implantation*. MEASURE propose un outil dédié à chacune de ces activités : l'*éditeur*, le *vérificateur* et l'application mobile. Ces outils sont construits autour des trois contributions présentées dans ce chapitre le MAML, et le EMAML, et dans le chapitre précédent le *MKQL*.
Ce chapitre présente en premier lieu le MAML. C'est un langage permettant la spécification de modèles d'applications par un architecte logiciel sans expertise dans la métrologie. Pour cela il propose un ensemble de termes permettant de modéliser une procédure de mesure sous la forme d'une séquence d'activités comprenant des opérations de relevé, de manipulation et de présentation de grandeurs. Ce langage est construit à partir des concepts, formalismes et outils proposés par l'environnement de métamodélisation Diagrammatic Predicate Framework (DPF). Le langage comporte des contraintes syntaxiques prévenant les erreurs de construction.
Dans un deuxième temps, l'application logicielle employée par l'utilisateur final est présentée. Cette application est configurable et adapte son comportement en fonction du modèle fonctionnel qui lui est donné. Le langage de description des modèles fonctionnels, le EMAML, est présenté. C'est un langage destiné à représenter une procédure de mesure d'un point de vue fonctionnel pour un système logiciel. Une procédure de mesure se retrouve réduite à un ensemble d'opérations de collecte, de manipulation et d'affichage de données séquencées. Cette dénaturation des grandeurs en simples données permet une gestion aisée des différentes opérations de manipulation, tant qu'une uniformisation est effectuée aux entrées et sorties de l'application. L'application logicielle est construite avec un framework permettant de construire le comportement de l'application. Ce framework est fondé sur une architecture d'agents. Chaque agent a un rôle simple. C'est leur collaboration qui permet de prendre en charge la gestion d'une séquence d'activités composée de relevés, de manipulations et de présenta-

tions de données. L'application contient également un constructeur permettant de générer la configuration correspondant au modèle d'implantation. Enfin, le chapitre se conclut en proposant deux transformations de modèles permettant d'intégrer l'éditeur de modèle, le vérificateur et l'application mobile dans la plateforme MEASURE. Cela permet d'automatiser les processus de vérification de la métrologie et d'implémentation de l'application mobile à partir du modèle de spécification.

Le **chapitre 6** illustre l'utilisation de la plateforme MEASURE en présentant le développement d'une application d'estimation du coût d'installation de panneaux solaires sur une toiture. L'application a été pensée avec Actimage GmbH pour permettre de faire avancer la réflexion sur la thématique de nos travaux. Ensuite, le chapitre présente l'adaptation de l'approche présentée dans ces travaux et son utilisation par Actimage GmbH sous la forme de la plateforme ActiNote 4.0^{TM}. Cette plateforme propose la description d'applications d'assistance aux interventions in situ. Le chapitre décrit les adaptations qui ont été apportées aux différents outils par Actimage pour proposer un système commercialisable.

Le **chapitre 7** conclut sur ces travaux et présente les différentes pistes à explorer pour rendre ses travaux plus accessibles à la communauté des développeurs.

Première partie

État de l’art

Chapitre 2

La science de la mesure

« *The mole is a quantity of substance. The new prefix "guaca" is defined such that one guacamole equals Avocado's Number.* »

G. David Byrne

Sommaire

Afin de comprendre, décrire et communiquer sur le monde, l'être humain a dû développer un vocabulaire adapté. La métrologie est employée dans toutes les disciplines, que ce soit les sciences dures (i.e. mécanique, physique, électronique), mais également les sciences humaines (i.e. intelligence, conscience) pour quantifier, jauger et raisonner. C'est elle qui caractérise les outils pour représenter les phénomènes observables. Pour cela elle définit les représentations possibles pour des propriétés observées et les relations qui les régissent.

Ce chapitre présente les concepts essentiels de la métrologie, la science de la mesure. Il met en avant l'utilisation de mesures pour communiquer et raisonner sur les phénomènes du monde observable. L'accent est mis sur l'importance de la métrologie et les règles employées par les experts pour certifier des procédures de mesures.

2.1 Abstraire le monde réel, normalisations et conséquences

Cette section présente un bref historique de la métrologie pour ensuite montrer les efforts réalisés pour normaliser le vocabulaire de la métrologie. Ces normes visent à simplifier la communication entre les experts du domaine. Leurs apparentes simplicités entraînent également un ensemble de conséquences négatives. En effet les résultats de mesures sont souvent manipulés comme de simples nombres. Les avantages et les inconvénients de ces normes sont détaillés ici.

2.1.1 Historique

La mesure est employée depuis les temps anciens [16]. L'action de mesurer est nommée mesurage. C'est une activité qui a pour but d'encoder des évènements empiriques en représentations formelles, simplifiant la communication de ces évènements, nommées grandeurs [89]. Un mesurage implique des processus complexes et est par nature multidisciplinaire. Fondé sur des comparaisons et des décomptes, un mesurage permet de quantifier la contenance, la taille ou encore le prix d'entités.

Auparavant, les comparaisons étaient fondées sur des parties du corps humain (pouce, coudée, pied) et des objets du quotidien (baril, amphore, graine de caroubier aussi appelée carat). Les références n'étant pas globales, certaines portaient le même nom, mais ne représentaient pas la même valeur d'un lieu à un autre. De plus, les subdivisions des références

étaient également basées sur des entités physiques. Ainsi il était rare que la conversion entre deux références soit une opération simple.

Par exemple, une coudée est divisée en six palmes. Une palme étant définie par quatre doigts. Il faudra 24 doigts pour une coudée. La conversion entre ces entités n'est donc pas aisée. De plus, une autre unité nommée coudée était employée : la coudée de maya. Elle était divisée en sept palmes, car utilisée pour les monuments religieux, sept étant un nombre sacré. Enfin, le doigt n'était pas identique dans chaque lieu. Dans ces conditions, il est impossible de connaître la taille exprimée par une longueur en coudées sans précisions sur le contexte.

Il faut garder à l'esprit que les grandeurs sont employées pour décrire les objets qui constituent le monde. Cette définition est importante pour pouvoir communiquer les propriétés caractéristiques de l'objet. Un plan d'architecte doit pouvoir être lu non seulement par son créateur, mais également par le maître d'oeuvre, son personnel et idéalement l'acheteur. Il faut donc que toutes ces personnes adoptent un vocabulaire commun et que les références employées soient partagées par tous.

2.1.2 Normalisation

Depuis la Révolution française, un effort conséquent a été réalisé afin d'uniformiser les notions de références. Pour garantir l'invariabilité des références, elles ont commencé à être définies à partir de phénomènes naturels fixes et reproductibles. Le premier mètre a été défini par rapport à la longueur du méridien terrestre. Cette définition est dite universelle puisque identique pour tous.

Cet effort a mené à la création du Joint Committee for Guides in Metrology (JCGM) qui a publié un vocabulaire unifié le Vocabulaire International de Métrologie (VIM) [89] ainsi que d'un ensemble de références internationales nommé le Système International (SI)[10]. Ces deux documents sont les références majeures constituant la métrologie actuelle.

La métrologie est la science des mesurages. Elle définit un ensemble de vocabulaires et de méthodes permettant d'encoder les évènements empiriques en abstractions formelles simples à communiquer et à comprendre. De plus, la manipulation de ces abstractions formelles permet d'en déduire d'autres qui sont également associées à des évènements empiriques. Les relations induites par la métrologie entre les domaines empirique et formel sont illustrées dans la figure 2.1. Cette figure met également en avant l'importance de la sémantique de la métrologie qui permet de contraindre les manipulations d'abstractions.

Le VIM est la référence définissant le vocabulaire international de la

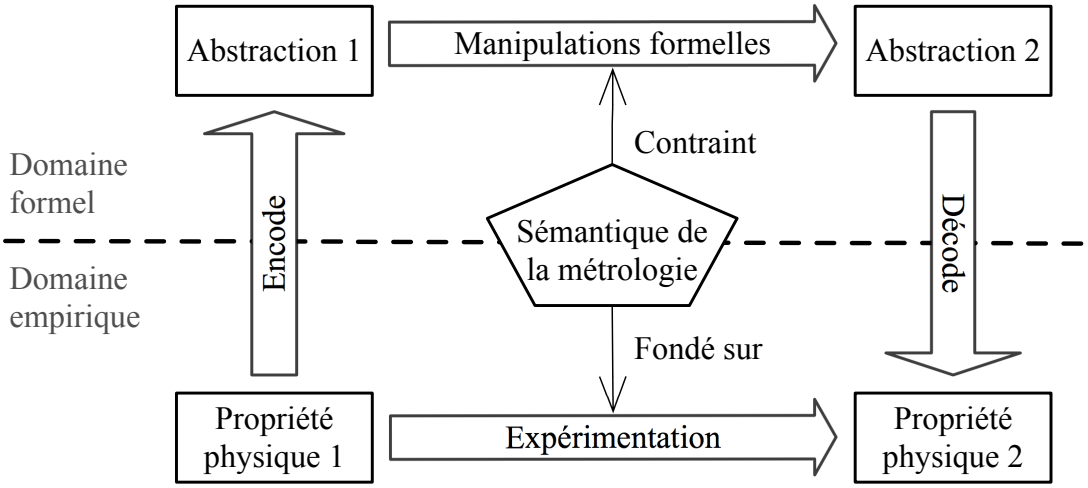

Figure 2.1 – Relations induites par la métrologie entre les domaines empirique et formel

métrologie [1]. Il comprend tous les termes que toute personne réalisant des mesurages devrait utiliser. Le modèle conceptuel du domaine présenté dans le chapitre 4 est déduit de ces termes.

Le VIM nomme l'abstraction formelle *grandeur*. Le but des grandeurs est de simplifier la communication d'évènements observés. Une grandeur est composée d'une *valeur numérique* et d'une *référence*. Pour simplifier le discours, nous utiliserons le terme *valeur d'une grandeur* pour parler de la valeur numérique d'une grandeur.

La référence d'une grandeur représente une quantité connue et figée de l'évènement soumis au mesurage. Partager cette référence permet de quantifier les évènements empiriques en définissant les relations entre la référence et l'évènement observé. Ainsi, pour que la grandeur soit facilement utilisable, il faut que la référence associée soit répandue et connue par tous ces utilisateurs. C'est pourquoi le SI a été proposé. Il définit un ensemble de références internationales nommées *unités*. Le SI assigne une référence pour chaque nature d'évènement quantifiable.

Chacune des unités du SI est définie par un étalon international unique. Des versions dérivées sont partagées au travers du globe et utilisées pour étalonner les instruments de mesure. Cela permet de proposer un vocabulaire unique pour identifier et comparer les grandeurs universellement. Le SI propose des subdivisions décimales des unités construites à partir du nom de l'unité et d'un *préfixe* (e.g. pico, centi, kilo) qui permettent d'expri-

1. Tous les termes propres à la métrologie abordés dans ce document sont définis par le VIM.

mer toutes les valeurs d'une grandeur avec une seule unité, et ce malgré l'écart entre les échelles dans les différents domaines (infiniment petit pour la physique des particules contre infiniment grand pour l'astrologie).

La valeur d'une grandeur est définie en comparant la référence à l'évènement empirique observé en utilisant un *instrument de mesure* avec la valeur affichée par un *instrument indicateur* . Généralement, il est possible d'appliquer des opérations de *conversion* à une grandeur pour changer sa référence en une autre compatible.

Un *mesurage* est l'application d'une *procédure de mesure* pour collecter une ou plusieurs valeurs de grandeurs. Une procédure de mesure est définie par l'association d'instruments de mesure, d'instruments indicateurs et de *fonctions de mesure*. Une fonction de mesure est une manipulation algorithmique de grandeurs qui implique une grandeur résultante en fonction de grandeurs sources. L'utilisation de fonctions de mesure permet la création de nouvelles grandeurs qui représentent également des propriétés physiques (figure 2.2). Cela permet de projeter et valider des propriétés du monde réel sans réaliser des expérimentations plus délicates. Par exemple un simple calcul permet de définir si un meuble peut être placé dans un espace donné quand l'expérimentation nécessiterait d'avoir le meuble à disposition. Un mesurage permet donc d'abstraire des évènements physiques pour en simplifier la communication et, en manipulant les abstractions, de prévoir certains évènements.

Le vocabulaire défini jusqu'ici est utilisé pour décrire les mesurages, les outils pour les réaliser, les abstractions et les méthodes pour les manipuler. La figure 2.2 présente un raffinement de la figure 2.1 utilisant les termes du VIM.

2.1.3 Conséquences

Le SI et le VIM étant des références internationales, ils permettent de simplifier la communication entre toutes personnes réalisant des mesurages.

Un expert de la métrologie est capable de définir une procédure de mesure, de spécifier les outils et fonctions de mesure nécessaires afin d'obtenir une grandeur. Il est capable de transmettre aisément la grandeur afin de l'utiliser dans d'autres traitements. De plus, puisque le vocabulaire est également unifié, il peut aussi transmettre la procédure de mesure afin de reproduire le mesurage dans des conditions différentes ou simplement de l'appliquer à une autre entité.

Cependant, le VIM est un outil qui n'est réellement employé que par les experts de la métrologie. Ce vocabulaire est généralement inconnu des

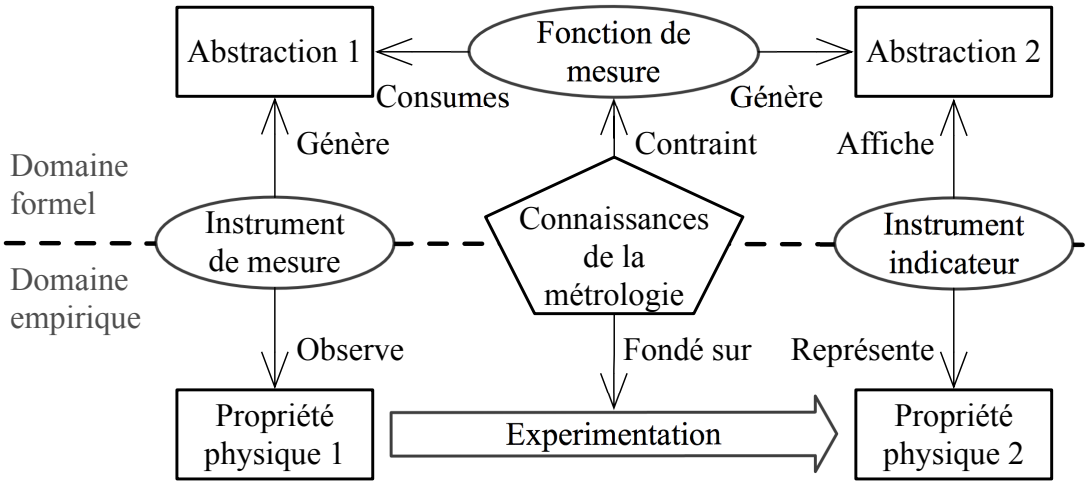

Figure 2.2 – Raffinement du cadre impliqué par la métrologie en prenant en compte le vocabulaire international

non experts qui utilisent un vocabulaire plus commun et plus ambigu. Par exemple, les termes de mesurage, grandeur et instrument de mesure sont communément appelés mesure, quantité et capteurs.

Les bases du SI sont bien mieux maîtrisés et communément employées par l'ensemble des personnes susceptibles de réaliser des mesurages. Le point fort du SI est de proposer des références internationales uniques qui ont été rapidement adoptées. Cependant, des unités hors du SI sont toujours employées, voire même autorisées par le SI [10]. Leurs utilisations peuvent être justifiées par :

— des raisons historiques, comme dans la navigation maritime ou aérienne avec les noeuds ou milles marins,

— un ancrage fort dans le vocabulaire commun, comme c'est le cas des minutes, litres et les tonnes,

— une commodité de notation, comme les unités atomiques, astronomiques ou basées sur des constantes naturelles,

— des équations simplifiées pour des domaines spécifiques, comme en électrodynamique avec l'utilisation d'un autre système d'unités, le Système d'unité Centimètre, Gramme, Seconde (CGS).

À cela s'ajoutent toutes les unités qu'emploient les pays anglo-saxons qui ont conservé le système impérial (e.g. mile, inch, degré Fahrenheit). La présence mondiale de ces pays et leur puissance économique engendrent une grande utilisation de ces unités.

Cette uniformisation incomplète implique que, naturellement les utilisateurs ont tendance à penser que les unités qu'ils emploient sont celles

employées par tous. Cela peut induire des erreurs dans des contextes internationaux. Prenons l'exemple de deux personnes parlant de la vitesse moyenne d'un conducteur sur un trajet. Elles pourraient, en langage commun, dire "*je pense qu'il était à 110*". Omettre l'unité est commun puisque les deux personnes savent que la thématique est la vitesse moyenne d'un véhicule. Mais même si l'orateur français est respectueux des limitations sur autoroutes, l'auditeur anglais pensera qu'il ne l'est pas. En effet, il entendra que la vitesse est en mile par heure, l'unité de vitesse anglaise et 110 $mile/h$ correspondent à 177 km/h. Bien qu'étant dans un contexte plus complexe, c'est une erreur d'unités équivalente qui est à l'origine du crash de la sonde *Mars Climate Orbiter*[118].

Outre les possibles problèmes de conversions, la représentation d'évènements empiriques en grandeurs a également tendance à être source d'erreurs. En effet, une grandeur est l'association d'une valeur numérique et d'une référence. L'utilisation d'une valeur numérique est tellement commune que les non-experts ont tendance à penser qu'ils manipulent des nombres au lieu d'abstractions de propriétés du monde réel. Cependant, les grandeurs n'ont pas forcément toutes les propriétés algébriques d'une simple valeur numérique.

Les figures 2.3 et 2.4 montrent deux de ces erreurs.

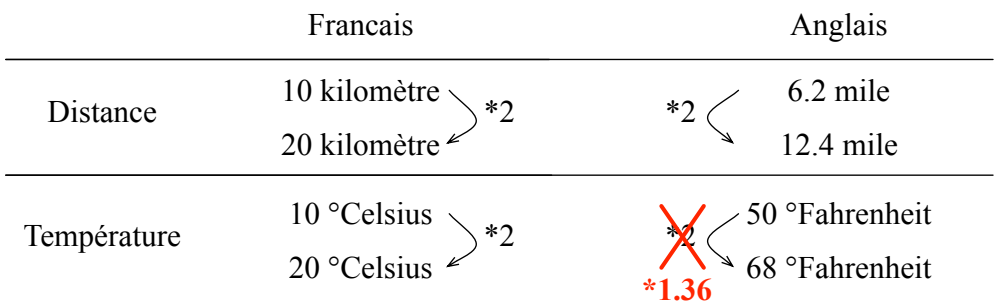

Figure 2.3 – Limites algébriques impliquées par des températures en °C ou °F

La figure 2.3 montre que la déclaration "*il fait deux fois plus chaud en mars qu'en mai*" est erronée. En effet, en fonction de l'unité employée le résultat n'est pas le même. Les zéros attribués aux degrés Celsius et Fahrenheit sont arbitraires (ne représentent pas l'absence de température) et différents. Le rapport entre deux températures ne représente donc pas une relation physique.

La figure 2.4 illustre une erreur d'inattention qui apparaît lorsque les grandeurs sont considérées comme de simples valeurs. Un véhicule parcourt deux trajets (A) et (B). Chaque trajet est coupé en deux intervalles identiques (A) en longueur ou (B) en temps. Le véhicule roule à une vitesse

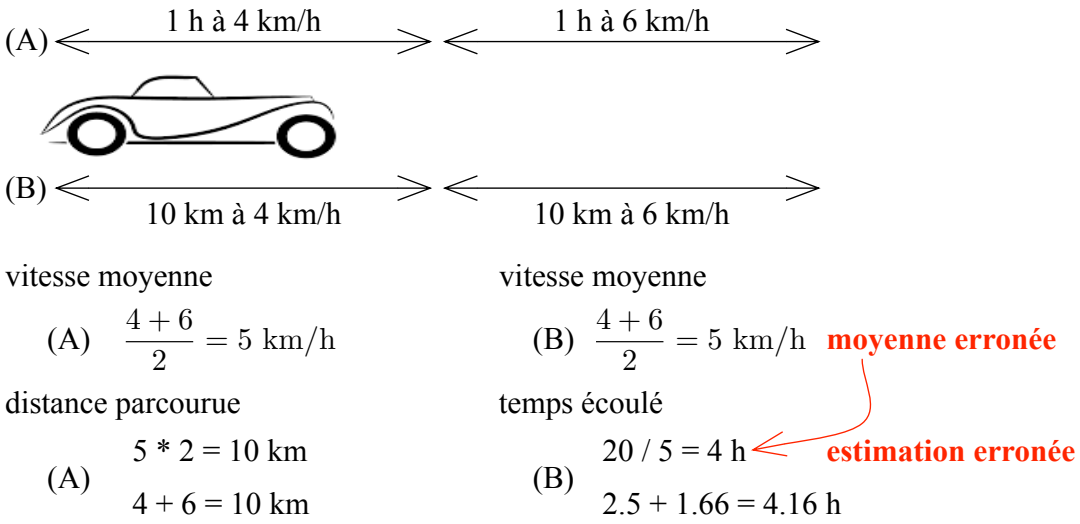

Figure 2.4 – Calcul et utilisation d'une vitesse moyenne correcte (A) et incorrecte (B)

différente sur chaque intervalle. La figure montre que, sur trajet découpé en intervalles identiques, calculer la vitesse moyenne en appliquant une moyenne algébrique est (A) possible pour des intervalles de temps identiques et (B) impossible pour des intervalles de longueurs identiques. Cela s'explique, car pour ces équations, la longueur et le temps sont respectivement numérateur et dénominateur. La somme de deux ratios aux dénominateurs différents n'est possible qu'en les uniformisant et c'est cette étape qui est oubliée dans le cas (B). Cette limitation peut être généralisée à toute grandeur dont la référence est définie par un ratio d'unités (e.g. vitesse, flux magnétique, force).

Cette section a présenté l'importance des travaux d'uniformisation qui ont été proposés afin de simplifier la communication dans des contextes de mesurages. Les limitations et erreurs communes induites par cette uniformisation ont également été mises en avant. La section suivante va présenter les différentes méthodes que la métrologie propose pour assurer la cohérence des abstractions avec les propriétés physiques qu'elles représentent pour garantir les interprétations.

2.2 Interpréter les grandeurs

Les instruments de mesure sont maintenant très répandus. Il est naturel d'employer des valeurs associées à des unités pour décrire des évènements empiriques. Cependant, nous avons vu que, cette description sous forme nu-

mérique des valeurs de grandeurs tend à induire les non-experts en erreur. En effet, les grandeurs sont des abstractions de propriétés du monde réel. Il ne faudrait donc pas les manipuler sans connaître les conséquences que cela implique sur les propriétés abstraites. Cette section présente les différentes règles qui composent les connaissances de la métrologie de la figure 2.2. Ces règles garantissent la cohérence des grandeurs issues de fonctions de mesure avec les propriétés physiques observées ou présumées.

2.2.1 Systèmes de grandeurs et analyse dimensionnelle

La métrologie définit un ensemble de *natures de grandeur* (e.g. énergie, flux magnétique, pression) pour catégoriser chaque type d'évènement empirique. Les *systèmes de grandeurs* définissent un ensemble de relations algébriques entre les natures de grandeurs pour répercuter les lois de la physique qui lient les évènements empiriques. Ces systèmes sont fondés sur un ensemble de *grandeurs de base* considérées indépendantes ainsi que sur un ensemble de *grandeurs dérivées* définies par des combinaisons algébriques des grandeurs de base. Le système international de grandeurs, ou International System of Quantities (ISQ) [63] est le système de référence. Il propose sept types de grandeurs de base : longueur, masse, temps, courant électrique, température thermodynamique, quantité de matière et intensité lumineuse. Toutes les grandeurs dérivées sont définies par des opérations de multiplications ou des divisions de ces sept grandeurs.

Le SI propose une unité pour chacune des grandeurs de base définies par l'ISQ et représente également toutes les autres unités par une combinaison de ces unités. Les préfixes permettent une mise à l'échelle de l'unité par rapport à l'élément observé. En plus, certaines unités ont des *noms spéciaux* qui permettent de simplifier leurs utilisations (e.g. newton, joule, watt). Les différents diagrammes proposés par le National Institute of Standards and Technology (NIST) [2] présentent les différents types de grandeurs de l'ISQ ainsi que les unités associées et les relations algébriques les liants. L'utilisation de systèmes de grandeurs différents permet généralement de simplifier les équations et les unités les plus communes dans ces systèmes, comme le CGS pour l'électromagnétisme ou la chimie.

Les systèmes de grandeurs sont utilisés, car ils simplifient la communication en réduisant le vocabulaire nécessaire pour représenter toutes les grandeurs et unités. Ce sont des modèles formels représentant des lois régissant la physique. Ils permettent donc de refléter les effets des lois de la physique sur les abstractions formelles au travers d'une algèbre simple.

2. http ://physics.nist.gov/cuu/Units/SIdiagram2.html

Cette algèbre est appelée analyse dimensionnelle et est décrite comme capable de vérifier l'homogénéité d'équations impliquant des grandeurs ou encore de les simplifier [4]. Chaque type de grandeur peut être représenté sous une forme canonique. Cette forme canonique est un vecteur dont la longueur est égale au nombre de grandeurs de base du système. Le vecteur est appelé *vecteur aux dimensions* puisque chacune des grandeurs de base est assimilée à une dimension du vecteur. Le vecteur aux dimensions associé au SI a donc la forme : *<longueur, masse, temps, courant électrique, température thermodynamique, quantité de matière, intensité lumineuse>* généralement noté $< \mathrm{L}, \mathrm{M}, \mathrm{T}, \mathrm{I}, \Theta, \mathrm{N}, \mathrm{J} >$.

L'analyse dimensionnelle implique de prendre la forme canonique de toutes les grandeurs impliquées dans le mesurage et de vérifier que les différentes manipulations sur ces grandeurs ne génèrent pas un résultat incohérent. Par exemple, dans un processus d'installation de panneaux solaires, il faudra pouvoir assurer que la toiture supportera le poids de l'installation. L'impact de l'installation sera sûrement donné en kg/m^2. Pour trouver la masse appliquée par une installation spécifique il faudra donc appliquer l'opération *masse = surface * impact* dont la représentation canonique est *<0,1,0> = <2,0,0> * <-2,1,0>* ce qui donne *<0,1,0> = <0,1,0>*. L'équation est homogène, ce qui garantit que les calculs sont corrects vis-à-vis de l'analyse dimensionnelle.

2.2.2 Échelles de mesures et grandeurs non physiques

L'analyse dimensionnelle empêche la combinaison de grandeurs sans relations physiques. La théorie représentationnelle de la mesure, ou Representational Theory of Measurement (RTM) propose de consolider les relations entre les évènements empiriques et leurs abstractions formelles. L'ensemble des opérations qui peuvent être appliquées à une grandeur devrait être contraint en prenant en compte la nature empirique de l'évènement qu'elle abstrait [70; 119]. L'ensemble d'opérations qui peuvent être appliquées à une grandeur est nommé *échelle de mesure*. L'échelle de mesure associée à une grandeur est définie par la connaissance de l'évènement empirique abstrait et de la procédure de mesure dont elle est issue [53]. En effet, l'ensemble des opérations est déduit de l'évènement observé afin de garantir que l'abstraction formelle encode bien une propriété du monde réel. Le tableau 2.1 présente les quatre échelles de mesure de base.

Ainsi, puisque les températures en degrés Celsius ou Fahrenheit sont associées à l'échelle intervalle, il est impossible de leur appliquer un ratio et l'erreur "il fait deux fois plus chaud" n'a plus lieu d'arriver.

La RTM étend les capacités d'abstractions des mesurages aux grandeurs

Échelle	Relations	Operations	Exemple
Nominale	Catégorisées	= (*est un*)	Genre, Couleurs
Ordinale	Ordonnées	$=, <$	QI, Notes scolaires
Intervalle	Équidistantes	$=, <, +$	Températures (°F, °C)
Ratio	Équidistantes et Zéro absolu	$=, <, +, *$	Masse, Longueur

Tableau 2.1 – Les quatre échelles de mesure de base. (inspiré de [119])

non physiques alors que les références en métrologie ne considèrent que les propriétés physiques. De plus, elle définit un ensemble de contraintes qui assurent que les relations illustrées dans la figure 2.2 liant les domaines empirique et formel . Une propriété physique peut être abstraite par une grandeur et une grandeur abstrait une propriété physique. Le fait que ces relations soient des homomorphismes implique que la manipulation des grandeurs est soumise aux contraintes du domaine empirique. Il est donc proscrit d'appliquer une fonction de mesure à une grandeur si cette fonction n'a pas de projection possible dans une expérimentation.

Cependant, il n'est pas aisé d'assigner une échelle de mesure à un évènement empirique et cette assignation dépend fortement du but du mesurage [76]. Prenons l'exemple d'une course. Les coureurs peuvent être différenciés par des mesures présentant des échelles de mesure différentes :

— Nominale avec le nom ou numéro des brassards,

— ordinale avec le classement à l'arrivée,

— intervalle avec l'attribution de note par rapport à la performance,

— ratio avec le temps à l'arrivée.

Chacune de ces grandeurs peut être employée pour différencier un concurrent spécifique de l'ensemble. Néanmoins en considérant deux groupes de coureurs, seule la connaissance du temps à l'arrivé autorise de définir exactement quel groupe a été le plus rapide pour terminer le parcours à l'aide d'une moyenne du temps de parcours.

Malgré cette difficulté, la RTM répond aux besoins de mesurages de grandeurs non physiques [40; 78].

2.2.3 Procédures et incertitudes de mesure

Tout mesurage implique une *incertitude de mesure* générée par la limite de précision des instruments et la combinaison des erreurs induites par la procédure de mesure [39]. Il est impossible de relever une grandeur parfaite. Bien que les incertitudes et leurs impacts sur l'évènement observé ne puissent pas être évités, elles peuvent être estimées et réduites [39; 88]. Le Guide pour l'expression de l'incertitude des mesurages ou Guide to the expression of Uncertainty in Measurement (GUM) est la référence internationale concernant l'estimation des incertitudes de mesures [88]. Il présente le vocabulaire et les méthodes pour les estimer et amoindrir leurs impacts.

Il faut noter que c'est le contexte d'utilisation de la grandeur qui définit les valeurs d'incertitudes acceptables ou non. Une valeur d'incertitude de $1mm$ sur la largeur d'un toit est rarement critique. La même incertitude sur l'épaisseur d'un cheveu n'est, par contre, pas acceptable. D'un autre côté, le prix d'un mètre ruban n'est pas comparable à celui d'un instrument capable de mesurer fidèlement la largeur d'un cheveu.

Une procédure de mesure spécifiquement définie est l'approche la plus fiable pour assurer de produire des grandeurs dont les valeurs d'incertitudes correspondent aux besoins du mesurage. Une telle procédure permettra d'assurer une valeur d'incertitude tout en optimisant les coûts impliqués par l'acquisition et l'utilisation des instruments de mesure nécessaires. Les diagrammes causes et effets, aussi appelés diagrammes d'Ishikawa [62], utilisés en gestion de qualité pour représenter graphiquement les causes liées à un effet, permettent de simplifier cette tâche en associant des branches d'entrées aux différentes sources d'incertitudes [9]. Ces sources sont généralement la méthode employée pour le mesurage, l'opérateur qui réalise le mesurage, les instruments de mesure, l'environnement dans lequel est réalisé le mesurage et l'entité soumise au mesurage [38].

Différentes solutions logicielles proposent d'estimer les valeurs d'incertitudes. Elles emploient plusieurs approches. Par exemple, elles peuvent utiliser un modèle mathématique [37] ou encore appliquer la loi de la propagation des incertitudes [52].

Ces solutions logicielles impliquent des connaissances approfondies de la métrologie et un ensemble précis de spécifications à respecter est préconisé [28]. L'estimation des incertitudes est de première importance, cependant, ces travaux n'abordent pas cette thématique. L'accent est mis sur la création d'applications de mesures conforme à la sémantique de la métrologie en considérant que le mesurage proposé par l'utilisateur permet de relever des grandeurs qui correspondent à ses besoins.

Les connaissances de la métrologie présentées sur la figure 2.2 assurent

que les grandeurs encodent bien des évènements du monde réel, et ce pour toutes étapes de la procédure de mesure. De ce fait, il est possible d'appeler l'ensemble de ces connaissances la sémantique de la métrologie. Cette sémantique englobe les deux approches de vérifications présentées. Premièrement, les systèmes de grandeurs et d'unités contraignent la manipulation des grandeurs en fonction des lois de la physique auxquelles elles sont sujettes. Deuxièmement, la RTM renforce ces contraintes en prenant en compte que la manipulation de grandeurs doit respecter les propriétés intrinsèques aux évènements observés. Grace à ces considérations, elle permet également de prendre en compte des grandeurs non physiques. Ces deux points sont vérifiés par la combinaison de l'analyse dimensionnelle et l'analyse des échelles de mesure de la RTM.

2.3 La métrologie et l'informatique

Cette section présente une vue d'ensemble des approches qui permettent la modélisation de grandeurs et de procédures de mesures dans les systèmes logiciels.

Différentes approches ont été employées pour encoder les concepts de la métrologie dans des systèmes logiciels : ontologies, modélisation, encodage des unités ou des dimensions, etc. Cependant, le problème majeur de ces implantations est le manque d'une normalisation des concepts proposés [42]. Ce manque a généré un écosystème de logiciels sans cohésions, chaque système proposant des interprétations et encodages différents des notions de la métrologie.

L'*Open Geospatial Consortium* propose la norme Observations and Measurements (O&M) [64]. Ce modèle propose d'employer des classifications pour définir l'ensemble des valeurs associé à une grandeur, physique ou non. O&M permet de modéliser les mesurages de n'importe quel domaine en décrivant des séquences d'observations. Un framework [14] est développé autour de O&M et ajoute les notions de procédure et un langage pour l'encodage de capteurs nommé SensorML. O&M est utilisé par le consortium pour la description de procédures de mesures précises impliquant tout type de protocoles de mesurage. Cependant, étant employée par des experts, aucune approche pour vérifier la cohérence des procédures de mesures n'est proposée.

Le Unified Code For Units of Measure (UCUM) est un système d'encodage conçu pour inclure toutes les unités employées communément [111]. Son but est de proposer un encodage non ambigu pour la communication électronique de grandeurs. Cet encodage permet de représenter toutes

unités physiques sous la forme d'une trame de sept caractères ASCII. Le UCUM est un langage capable d'exprimer toutes les unités, même préfixées. De plus il ne présente pas de conflit dans les expressions d'unités du moment que les unités hors du SI ne sont pas préfixées. C'est donc une solution viable pour représenter les grandeurs comme des structures composées d'un nombre et d'une unité décrite par le UCUM. Malgré l'intérêt que présente l'encodage du UCUM pour la communication entre machines, il n'est dédié qu'à cette tâche et malheureusement n'est que rarement employé par les instruments de mesure communicants. Le UCUM ne propose aucune sémantique de la métrologie. Ainsi il ne devrait être employé qu'au niveau des interfaces de communication des systèmes logiciels l'employant.

Dans l'ingénierie des connaissances, une ontologie spécifie l'ensemble des concepts et termes d'un domaine tout en explicitant les relations les liants [50]. Les grandeurs permettent d'encoder la connaissance que l'on a d'un objet. Cet encodage est réalisé à l'aide d'un vocabulaire et de règles normalisés. Il est donc logique de trouver des ontologies pour représenter les grandeurs et les unités.

Quantities, Units, Dimensions and data Types (QUDT) [58] est une ontologie dédiée à la représentation de systèmes de grandeurs et d'unités physiques. Elle est capable de définir la dimension de n'importe quelle unité et de déduire le type de grandeur résultant de manipulations de grandeurs. QUDT propose une base réutilisable dans n'importe quel logiciel pour prendre en charge les grandeurs, les unités et leurs manipulations. Cependant, cette ontologie n'est pas capable de différencier des grandeurs de types différents mais présentant des dimensions identiques. Cela apparaît par exemple entre un couple mécanique et une énergie ou encore entre une grandeur sans dimension et un angle.

Ontology of units of Measure and related concepts (OM) est une ontologie qui modélise les concepts de la métrologie [99]. Son but est l'ajout de sémantique issue de la métrologie à des systèmes employant des mesurages. OM propose un service web pour appliquer l'analyse dimensionnelle sur des équations. Elle propose également de prendre en compte les domaines d'application. En définissant les unités usuelles pour un domaine donné, elle réduit le champ lexical des unités employées et évite de confondre des unités présentant des dimensions identiques et des sémantiques différentes. En effet, en sachant que le domaine d'application est le calcul de forces rotatives, OM peut retirer les unités concernant le domaine de l'énergie. OM propose d'étendre les capacités d'outils couramment employés par les chercheurs (e.g. Microsoft Excel^{TM}, Matlab^{TM}), en y ajoutant les concepts de la métrologie. Cependant, l'implantation de OM est basée sur une interprétation des échelles qui est différente de celle de la RTM [99]. OM n'est

donc pas capable de représenter des grandeurs non physiques.

Des approches alternatives étendent les capacités des langages de programmation classiques avec les notions de grandeur, d'unités et d'analyse dimensionnelles. Des travaux ont été réalisés pour employer les capacités du préprocesseur embarqué par le compilateur de C++ [23] et la librairie Boost propose une implantation relativement récente de ce concept [112]. L'idée est d'inclure les unités des grandeurs dans le code sous la forme de macros. La présence de grandeurs directement dans le code source facilite sa compréhension et responsabilise son rédacteur, car les entités manipulées ne sont plus simplement des valeurs numériques, mais des grandeurs. Cette approche évite la plupart des erreurs liées aux dimensions qui peuvent apparaître dans le code source d'une application de mesure. Cependant elle est fondée sur l'analyse dimensionnelle proposée par le SI. Elle ne considère pas les systèmes de grandeurs différents. De plus elle ne prend pas en compte la RTM et les grandeurs non physiques.

Enfin, des langages et outils spécialisés pour la modélisation de systèmes physiques existent [1; 84]. Ils permettent une modélisation fidèle des grandeurs et de leurs manipulations. Cependant, ils ne sont pas conçus pour être utilisés dans des processus de développement logiciels, mais pour la modélisation et la simulation de systèmes physiques.

Le tableau 2.2 montre une comparaison des différentes approches.

	O&M	**UCUM**	**QUDT**	**OM**	**Preprocesseurs**
Unités	toutes	physiques	physiques	physiques	physiques
Procédure de mesure	complexe	aucune	basique	basique	basique
Analyse dimensionnelle	non	non	oui	oui	oui
Analyse des échelles	non	non	non	non	non
Connaissances requises	expert	basique	basique	basique	basique
Utilisation	systèmes employant des capteurs complexes	encodage d'unité et communication	vérification	vérification et aide aux experts	vérification de code source

Tableau 2.2 – Comparaison des systèmes permettant de modéliser des grandeurs ou des procédures de mesures

2.4 Conclusion

La métrologie est la science qui permet de définir les approches et les règles utilisées pour abstraire des évènements du monde réel en grandeurs, des abstractions formelles. Les grandeurs simplifient la communication employant des références connues de tous pour comparer les évènements de mêmes natures.

La sémantique de la métrologie, en plus de proposer un vocabulaire international, définit un ensemble de contraintes assurant la cohérence entre les grandeurs et les évènements empiriques. Ces contraintes peuvent être vérifiées en appliquant l'analyse dimensionnelle des systèmes de grandeurs et l'analyse des échelles de mesure de la RTM.

Différents systèmes logiciels proposent de modéliser des grandeurs ou des procédures de mesure. À cause d'un manque de formalisme dans le domaine de l'informatique, aucune normalisation n'est proposée. Ainsi, les systèmes proposent des points de vue, des encodages et des formalismes différents et génèrent un écosystème sans cohésion.

Les travaux présentés ultérieurement emploient trois points de vue différents sur ce qu'est une procédure de mesure en fonction du contexte d'utilisation (modélisation d'application, vérification de procédures de mesure, implantation d'application). Le chapitre suivant présente les différentes approches et outils, issus du monde du logiciel, employés pour réaliser ces travaux.

Chapitre 3

Génie Logiciel

« *C'est seulement pour ceux qui persévèrent après que tout semble perdu, que l'espoir luit à nouveau.* »

Gilbert Keith Chesterton

Sommaire

Ce chapitre présente différentes approches du génie logiciel. Celles-ci répondent aux différents besoins d'un processus de développement dans le contexte de ces travaux.

Cette présentation commence par une description des approches de représentations de connaissances employées pour créer des systèmes capables de raisonner. Deux approches sont étudiées, les logiques descriptives et la programmation logique. Elles peuvent être utilisées afin de modéliser et de raisonner sur un domaine pour générer un système expert.

Ensuite ce sont trois paradigmes employés pour l'implémentation de systèmes logiciels qui sont présentés. Ces paradigmes sont les objets, les composants et les agents. Ils permettent de simplifier l'implantation des systèmes logiciels qui seront développés dans ces travaux.

La section suivante présente la modélisation de systèmes. Elle permet de décrire des descriptions, formelles ou non, des structures, comportements et communications d'un système. Ces descriptions sont employées pour communiquer avec les différentes parties intervenant dans le développement d'un logiciel. Les langages génériques et les langages dédiés sont présentés.

Enfin, l'Ingénierie Dirigée par les Modèles, une approche qui permet d'utiliser les modèles comme unités descriptives d'implantation est abordée. Cette approche permet d'adapter les points de vue sur le système afin de permettre l'interopérabilité des approches présentées tout en profitant de leurs avantages. Les notions de métamodélisation, utilisées pour la création de langages de modélisation, et de transformation de modèles, utilisées pour l'adaptation du modèle en code d'implémentation, de vérification ou encore pour sa mise à jour sont décrites.

3.1 Représentation des connaissances et mécanismes de raisonnement

Une représentation des connaissances, ou Knowledge Representation (KR) permet d'encoder les connaissances propres à un domaine sous forme numérique. Cet encodage permet de partager ces connaissances et de raisonner dans le domaine comme le ferait un être humain [32]. L'approche la plus utilisée est la logique mathématique [25].

Les langages les plus usités (Java, C++, C#) pour le développement de systèmes logiciels sont des langages impératifs [36; 121]. Cette désignation implique que le logiciel est décrit sous la forme d'instructions qui seront suivies à la lettre lors de l'exécution. Le développeur décrit donc les structures

1	ensoleillé	élément
2	juillet	élément
3	sec	élément
4	couleur(soleil, jaune)	relation
5	mois(juillet,7)	relation
6	**si** ensoleillé **et** juillet **alors** chaud	relation complexe
7	**si** juillet **et** humide **alors** orageux	relation complexe
a	le temps est-il ensoleillé ?	requête
b	est ce qu'il fait chaud ?	requête
c	le temps est-il orageux ?	requête

Tableau 3.1 – Exemple d'une représentation des connaissances basiques de météo.

de contrôle et comment elles interagissent. Les langages employés pour définir les KR sont généralement définis comme des *langages déclaratifs* [74]. Ces langages décrivent le système sans s'attarder sur l'utilisation qui en sera faite. Le développeur ne décrit plus une solution mais un problème.

Une KR contient des éléments et des relations liant les éléments. De ce point de vue, elles sont identiques aux bases de données classiques (relationnelles) [101]. Elles contiennent également des relations *complexes* qui permettent de lier des éléments et des relations. Les langages de description des KR sont liés à des sémantiques fortes dérivées de la logique. Cela permet à la KR de pouvoir répertorier les entités qui la composent mais également, une fois associée à un moteur d'inférence, de raisonner sur l'ensemble de ces entités en déduisant des relations qui ne sont pas expressément décrites [90]. Ces approches sont à l'origine de l'essor des premières Intelligences Artificielles utilisant les langages LISP ou Prolog [18; 32; 65] et continuent à être employées.

Le tableau 3.1 montre un example d'une KR dans le domaine de la météo. Elle est composée de trois éléments, de deux relations et deux relations complexes.

La réponse à la requête (a) est trouvée directement dans la KR avec l'élément (1) et est donc vraie. Les requêtes (b) et (c) nécessitent de pouvoir raisonner sur les connaissances pour répondre. La figure 3.1 montre le raisonnement qui permet de déduire qu'il fait chaud.

Il est donc important de noter qu'une KR n'est pas une représentation parfaite des connaissances du domaine représenté. Une personne développera une KR qui représente ses connaissances à l'aide de son vocabulaire et qui servira de support à ses raisonnements [32]. La représentation est donc une projection des connaissances spécifiques que le développeur possède sur le système original.

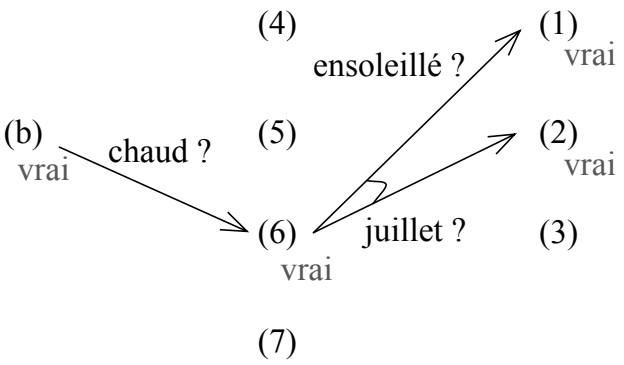

Figure 3.1 – Raisonnement permettant d'inférer qu'il fait chaud en utilisant la KR de météo.

Ainsi, ce ne sont pas tous les éléments de l'original qui sont représentés. Un raisonnement issu d'une KR donc peut induire un résultat qui ne correspondra pas aux propriétés réelles. Il est donc important de connaitre non seulement les capacités de la représentation mais également ses limites.

La requête (c) de la table 3.1 permet d'illustrer ces propos. Contrairement au raisonnement que ferait un humain, il n'est pas possible de répondre avec certitude à la requête (c). En effet, *ensoleillé*, *juillet* et *sec* sont des éléments de la base. Bien que la langue française implique que *sec* et *humide* soient deux contraires, rien ne le spécifie dans la KR. Ne connaissant pas la valeur pour *humide*, il est impossible de déterminer une réponse pour (7). De plus, le raisonnement pour la question (b) apporte une réponse inexacte s'il est effectué au pôle nord.

3.1.1 Systèmes experts

Une KR ne contient pas l'ensemble des connaissances d'un domaine. Elle encode les connaissances sous la forme d'entités et de relations. Il est ensuite possible, à l'aide de mécanismes d'inférence, de raisonner sur l'ensemble des éléments la constituant.

L'association d'une KR et d'un moteur d'inférence est souvent appelé système expert [65]. Les systèmes experts sont des systèmes d'intelligence artificielle. Le but de ces systèmes étant de déduire une réponse à une requête d'entrée, en raisonnant sur les connaissances d'un domaine. Ces systèmes sont utilisés depuis les années 80 et sont encore utilisés aujourd'hui, particulièrement avec l'émergence des ontologies (voir section 3.1.3).

Ces systèmes emploient deux types de connaissances [35] :

— **connaissances factuelles** : Connaissances axiomatiques accessibles directement dans la KR. Ces connaissances sont communément parta-

gées et peuvent être trouvées dans des manuels ou journaux spécialisés,

— **connaissances heuristiques** : Connaissances du domaine considérées comme bonnes pratiques, jugements ou encore raisonnement. Elles comprennent les décisions réfléchies dans le domaine.

Un système expert est généralement constitué de deux parties : une représentation des connaissances et un moteur d'inférence. La figure 3.2 représente un système expert ainsi que les deux types de connaissances qui peuvent être employées pour déduire des réponses.

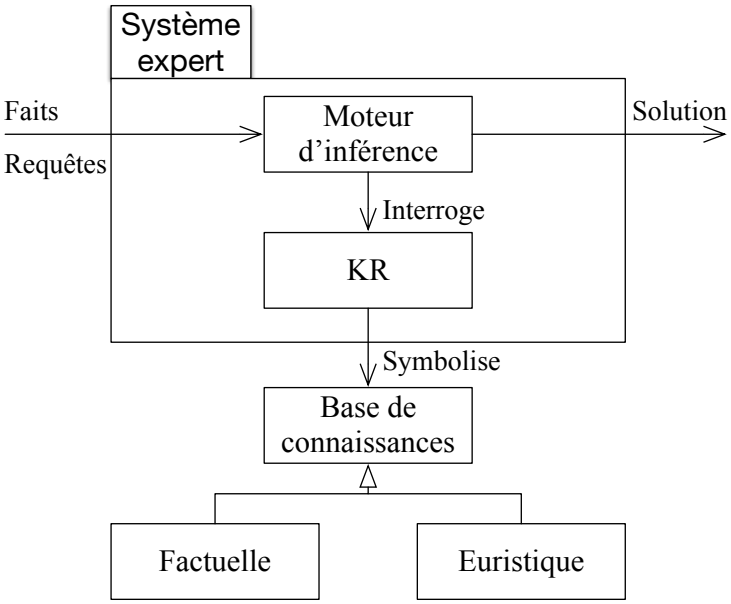

Figure 3.2 – Combinaison d'une représentation des connaissances et d'un moteur d'inférence dans un système expert.

Différents algorithmes d'inférence sont utilisés dans les systèmes experts. Les plus représentés et les plus simples sont les algorithmes de chainage avant ou arrière [22].

3.1.2 Syntaxes et sémantiques

La plupart des KR sont décrites en utilisant la syntaxe des calculs des prédicats aussi nommée logique du premier ordre, ou First Order Logic (FOL) [25; 97]. La FOL est une formalisation de la logique mathématique.

Elle propose une syntaxe permettant de décrire le raisonnement humain proche de l'expression d'un raisonnement par le langage naturel. Une

phrase telle que "*il existe un homme nommé Socrate*" aura en équivalent FOL ; "*il existe X tel que X se nomme Socrate et X est un homme*".

En plus de son expressivité, ce formalisme est construit autour des notions de démonstration. Il permet donc de définir la véracité d'un énoncé à partir des éléments le constituant. On pourra donc vérifier que Socrate est bien un homme.

Syntaxe

La syntaxe de la FOL se compose de termes, de prédicats, de connecteurs logiques et de quantificateurs.

Les *termes* représentent des éléments du domaine décrit. Il en existe trois sortes

1. une constante représente un élément spécifique : jack, jaune, a, 13,
2. une variable représente un ou plusieurs éléments qui répondent au contexte : X, Jack, Value,
3. une fonction définit un élément à partir des éléments utilisés en arguments. Elles peuvent être vue comme des constructeurs pour des éléments qui ont des attributs : bureau(jack), +(x,1).

Dans ce document, les constantes seront représentées par un nom commençant par une minuscule et les variables par une majuscule.

Les *prédicats*, également appelés atomes, représentent les relations entre des ensembles de termes. Ils sont constitués d'un symbole de prédicat et d'un ensemble de termes : mois(juillet,7), >(x,10), P, Q(r). Les prédicats sont équivalents à des fonctions qui assignent la valeur vrai pour chaque vecteur de termes qui remplissent la relation ; et faux pour chaque vecteur de termes qui ne la satisfont pas. Par exemple, le prédicat Q(r) associera la valeur vrai à toute réponse r qui répond à la question Q. P est un prédicat sans paramètre. Il peut être vu comme une variable à laquelle la valeur vrai ou faux sera associée par le domaine. Les prédicats sont donc des formules logiques définissant les vecteurs d'éléments soumis ou non à une relation.

En appliquant des *connecteurs* sur des formules il est possible de construire des formules dites complexes. Il existe cinq connecteurs, illustrés ci-après, avec A et B des formules :

1. la négation logique $\neg A$ est vraie si A est fausse,
2. la conjonction $A \wedge B$ est vraie si A et B sont vraies,
3. la disjonction $A \vee B$ est vraie si A ou B est vraie,
4. l'équivalence $A \Leftrightarrow B$ est vraie si A et B sont fausses ou sont vraies,
5. l'implication $A \Rightarrow B$ est vraie si A est fausse ou B est vraie.

Les deux derniers connecteurs permettent d'exprimer les règles qui régissent le domaine décrit. Ce sont ces règles qui permettent de raisonner sur l'ensemble des éléments énoncés.

La syntaxe propose deux *quantificateurs* illustrés ci-après avec X une variable :

1. le quantificateur existentiel $\exists X\ florent(X) \wedge doctorant(X)$ est "vrai" s'il existe un élément substituable à X qui se nomme Florent et qui est doctorant.
2. le quantificateur universel $\forall X doctorant(X) \Rightarrow étudiant(X)$ implique que *si X est doctorant alors X est étudiant* est "vrai" pour tout choix attribuable à X

Les quantificateurs permettent de restreindre l'ensemble des éléments qui peuvent être attribués à une variable. Une variable associée à un quantificateur est dite *liée ;* les autres variables sont dites *libres*. Si toutes les variables d'une formule sont quantifiées, alors elle est nommée *phrase*. Il est possible de définir la véracité d'une phrase.

La table 3.2 montre trois formules utilisant le même prédicat et montre les différentes interprétations qui en sont déduites.

1	$=(2X,X)$	indéfinie, X peut être tout élément
2	$\forall X\ entier(X) \wedge =(2X,X)$	faux, X peut être tout entier
3	$\exists X\ entier(X) \wedge =(2X,X)$	vrai, X peut être 0

Tableau 3.2 – Différentes interprétations pour un prédicat avec des variables liées ou libres.

La figure 3.3 représente la syntaxe de la FOL sous la forme d'un diagramme de classe. Toute formule exprimable à l'aide de ces éléments est une formule FOL correcte au niveau syntaxique.

Sémantique

La syntaxe de la FOL permettant de représenter les éléments et relations d'un domaine à l'aide de formules logiques a été présentée. Il faut maintenant associer une sémantique, un sens, aux formules.

L'*interprétation* d'une formule permet d'associer des objets du domaine aux éléments syntaxiques (termes et prédicats) qui la composent. C'est l'interprétation qui déterminera la véracité des formules. Prenons la formule $couleur(soleil, vert)$. Cette formule est fausse si l'interprétation associe le corps céleste nommé soleil à $soleil$ et la couleur verte à $vert$. Mais elle peut être vrai si l'interprétation associe le soleil dessiné par un enfant à $soleil$.

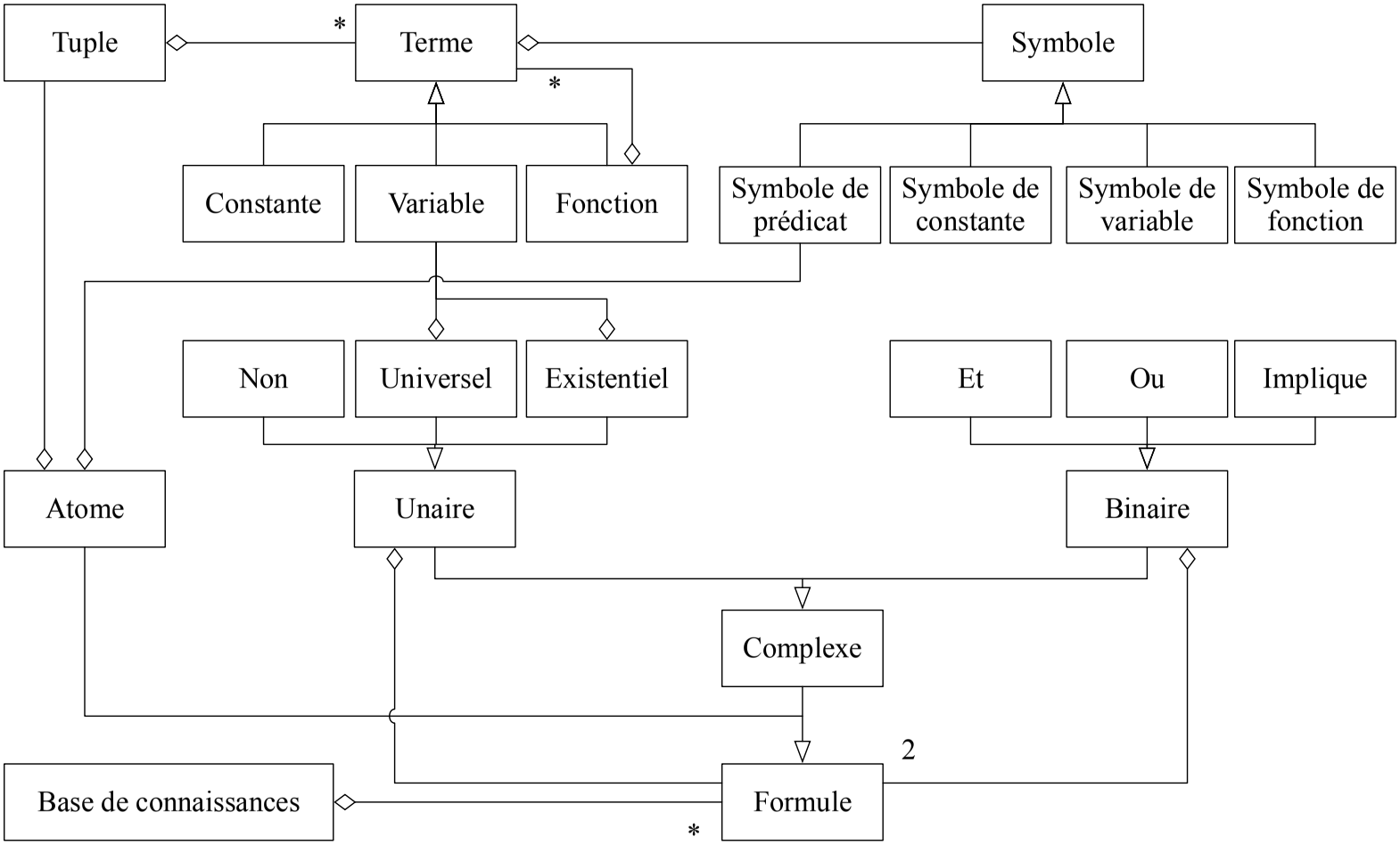

Figure 3.3 – Description de la syntaxe de la logique du premier ordre sous la forme d'un diagramme de classes.

C'est donc le domaine et l'interprétation qui en est faite (figure 3.4) qui permettent de déterminer la véracité d'une formule. Une interprétation qui assigne la valeur "vrai" d'une formule est appelée un *modèle* pour la formule.

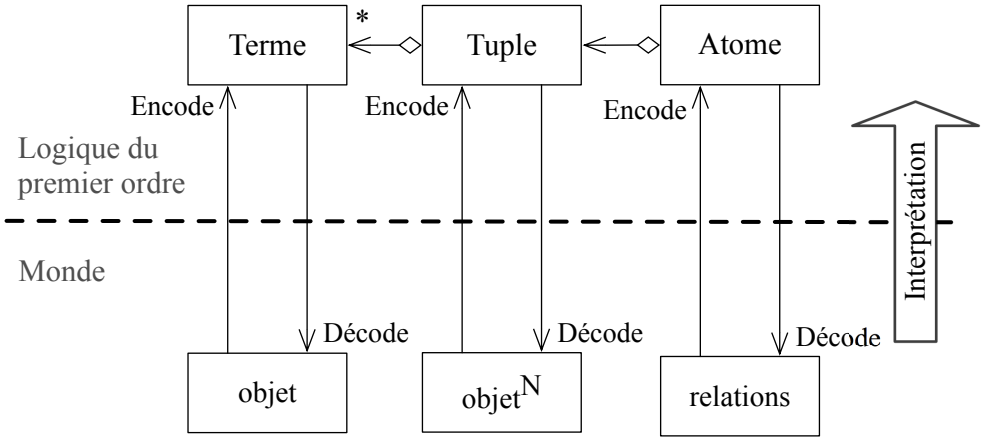

Figure 3.4 – Relations impliquées par l'interprétation entre le domaine et la syntaxe de la FOL.

Il est délicat de définir qu'une formule F est une conséquence logique d'un ensemble de formules E. En effet ce n'est le cas que si F est vraie, si toutes les interprétations satisfaisant toutes les formules E sont aussi des modèles pour F [41]. Ce qui implique d'étudier toutes les interprétations, qui peuvent être nombreuses.

Une autre approche, plus efficace à calculer, utilise des règles d'inférences (modus ponens, modus tollens, généralisation) [25] qui réécrivent les formules et contraignent leur analyse. Par exemple, en imposant que F est la dernière formule d'une séquence de formules S ; S étant la concaténation de l'ensemble de formule E et de F ; on pourra prouver que F est déductible de S en considérant que toute formule de la séquence satisfait une des trois conditions suivantes :

1. la formule est toujours satisfaite,
2. la formule est un axiome (un fait avéré dans l'analyse),
3. la formule est dérivée des formules qui la précèdent dans S en utilisant une règle d'inférence.

Ces approches optimisent le calcul et sont donc particulièrement utiles pour permettre de créer des outils logiciels capables de raisonner. Il existe un grand nombre d'outils capables de raisonner sur la FOL[29].

La figure 3.5 présente une KR permettant de représenter une généalogie. Le tableau 3.3 montre l'ensemble des formules la constituant.

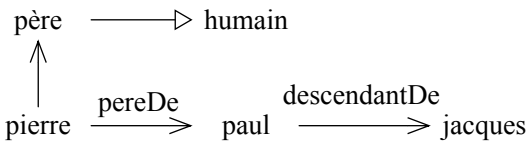

Figure 3.5 – Représentation des connaissances d'une généalogie.

1	$pere(pierre)$
2	$pereDe(pierre, paul)$
3	$descendantDe(paul, jacques)$
4	$\forall$ X ($descendantDe$(X,Y) $\vee$ $pereDe$(X,Y) $\Rightarrow$ $pere$(X))
5	$\forall$ X ($pere$(X) $\Rightarrow$ $humain$(X))
6	$\forall$ X $\forall$ Y ($pereDe$(X,Y) $\Rightarrow$ $descendantDe$(X))
7	$\forall$ X $\forall$ Y $\exists$ Z ($pereDe$(X,Z) $\wedge$ $descendantDe$(Z,Y) $\Rightarrow$ $descendantDe$(X,Y)

Tableau 3.3 – Formules FOL pour la représentation des connaissances d'une généalogie.

Cette KR permet de définir que Paul est le descendant de Jacques et de Pierre, que Pierre et Jacques sont des pères et que Pierre et Jacques sont des humains.

3.1.3 Ontologies et Prolog

Nous avons étudié deux approches communément employées pour décrire des représentations des connaissances et raisonner. Ce sont la logique descriptive [5] employée dans les ontologies [79] avec l'ensemble de langage OWL [90; 127] et la programmation logique [75; 132] utilisée dans les moteurs de règles d'intelligences artificielles [49] avec le langage Prolog [120; 133].

Logique descriptive

Une ontologie est une représentation des connaissances d'un domaine. Depuis quelques années, l'avènement du web sémantique [8] a entrainé un nombre croissant de travaux mettant à disposition des ontologies représentant les connaissances de multiples domaines.

Les logiques de description proposent trois types d'entités : les *concepts* qui sont des classes d'individus, les *rôles* qui sont des relations entre individus et les *individus* qui sont des instances des concepts.

Une KR décrite avec les logiques de description est structurée en deux parties, la *ABox* qui exprime les axiomes auxquels sont soumis les individus

et la *TBox* qui définit la structure de l'ontologie à l'aide des concepts et rôles.

Plusieurs familles de logiques de description existent. Elles proposent toutes un ensemble d'éléments nommés constructeurs. Les constructeurs permettent de décrire les entités d'un domaine avec une syntaxe plus ou moins proche de la FOL.

OWL ou Web Semantic Langage est le standard proposé par le W3C pour représenter les ontologies. C'est un ensemble de langages dérivé du langage de description de graphes Resource Description Framework Schema (RDFS) et les logiques descriptives [2].

RDFS définit un ensemble de classes de base pour décrire formellement et typer les ressources publiées sur le web et décrire leurs relations à l'aide de triplets de la forme $< sujet, prédicat, objet >$. Il permet donc la description de termes et de prédicats des FOL

OWL augmente d'un niveau les relations exprimables en ajoutant des constructeurs équivalents aux connecteurs et quantificateurs de la FOL ainsi que des constructeurs plus expressifs comme la restriction de cardinalité ou des concepts globaux permettant d'adresser tous les individus ou aucun.

Le tableau 3.4 montre la KR de la figure 3.5 décrite en logique de description.

1	H*uman* *is*A C*lass*
2	P*ère* *sub*C*lass*O*f* H*uman*
3	*père*D*e* *is*A P*roperty*
4	P*ère* *père*D*e* H*uman*
5	*descendant*D*e* *is*A T*ransitive*P*roperty*
6	*descendant*D*e* *inverse*O*f* *père*D*e*
7	H*uman* *descendant*D*e* P*ère*
a	P*ère*(*pierre*)
b	*pierre* *père*D*e* *paul*)
c	*paul* *descendant*D*e* *jacques*

Tableau 3.4 – Description OWL pour la représentation des connaissances d'une généalogie. La partie supérieure définit les classes et propriétés (TBox). La partie inférieure définit les instances (Abox)

Le langage est dérivé en plusieurs sous-langages aux capacités d'expressions et complexités d'inférences croissantes : OWL Lite, OWL DL et OWL Full [71]. OWL Lite propose un ensemble réduit de constructeurs assurant sa compatibilité avec RDFS et que toute démonstration est décidable. OWL

DL propose un ensemble syntaxique décidable. Il n'est cependant pas entièrement compatible avec RDFS. OWL Full offre une expressivité maximale, une compatibilité avec RDFS mais ne garantit pas la décidabilité des démonstrations. Le choix du langage est à faire en fonction de l'utilisation de l'ontologie.

Il est à noter que le développement d'une ontologie implique d'employer plusieurs langages (pour la description des ABox et TBox) et impose des environnements de développement ainsi que des formalismes contraignants.

Les ontologies permettent un accès aux résultats d'un développement simplement au travers d'internet et permettent une grande expressivité. Les logiques descriptives imposent que le raisonnement soit monotone. La KR ne peut donc pas être modifiée par le raisonnement. Les raisonnements réalisés utilisent l'hypothèse du monde ouvert. Cette hypothèse implique que toute formule est considérée vraie tant que l'on ne peut pas prouver qu'elle est fausse.

Programmation logique

La programmation logique quant à elle propose de représenter l'ensemble d'une KR sous la forme de règles et de faits. Les faits représentent les axiomes du domaine et les règles, les règles logiques qui régissent le domaine.

C'est le langage Prolog (Programmation logique) qui est le plus utilisé dans ce domaine. Pour simplifier et accélérer la démonstration des expressions de la FOL, prolog propose de décrire les formules complexes sous la forme d'un ensemble de clauses de Horn [60]. Une clause de Horn est une conjonction de clauses qui est composée au plus d'un seul littéral positif et d'un nombre fini de littéraux négatifs. Un littéral est un prédicat ou la négation d'un prédicat.

Ainsi, la formule complexe $A1 \wedge A2 \wedge A3 \Rightarrow B$ composée des formules A1, A2, A3 et B peut être réécrite sous la forme $(\neg A1 \vee \neg A2 \vee \neg A3) \vee B$. B sera alors démontrée en démontrant consécutivement A1, A2 et A3.

Pour Prolog, il existe trois sortes de clauses.

1. les faits qui représentent les axiomes du domaine. Un fait est une clause ne contenant aucun littéral négatif.
2. les règles qui représentent les relations logiques du domaine. Une règle est une clause avec un littéral positif et un nombre fini de littéraux négatifs,
3. les buts qui représentent une requête de démonstration qui sont généralement les entrées d'un système expert. Un but est une clause avec

uniquement des littéraux négatifs.

Le tableau 3.5 montre la KR de la figure 3.5 décrite en Prolog.

1	$pere(pierre).$
2	$pere(\mathrm{X}) :- pereDe(\mathrm{X}, \_).$
3	$pere(\mathrm{X}) :- descendantDe(\_,\mathrm{X}).$
4	$pereDe(pierre, paul).$
5	$humain(\mathrm{X}) :- pere(\mathrm{X}).$
6	$descendantDe(paul, jacques).$
7	$descendantDe(\mathrm{X},\mathrm{Y}) :- pereDe(\mathrm{Y},\mathrm{X}).$
8	$descendantDe(\mathrm{X},\mathrm{Z}) :- pereDe(\mathrm{X},\mathrm{Y}), descendantDe(\mathrm{Y},\mathrm{Z}).$

Tableau 3.5 – Règles Prolog pour la représentation des connaissances d'une généalogie.

La distribution la plus utilisée de Prolog est SWI-Prolog. C'est une distribution qui a été construite dans l'optique de créer des systèmes experts impliquant des besoins d'interactions et de raisonnement élevés. La communauté d'utilisateurs contribue à l'élaboration de librairies de règles permettant d'augmenter les capacités du langage tout en simplifiant son utilisation [133].

SWI-Prolog propose un raisonnement qui est partiellement non monotone. Il est donc possible de modifier la KR pendant le raisonnement en ajoutant ou retirant des clauses temporaires. Les raisonnements réalisés utilisent l'hypothèse du monde fermé. Toute formule est considérée fausse à moins que le contraire ne soit démontré. SWI-Prolog permet de mélanger des approches déclaratives et impératives. Ce qui permet de manipuler la démonstration de requêtes en manipulant l'algorithme de raisonnement. Par exemple en réduisant le nombre de règles à analyser [86].

Ces travaux utilisent SWI-Prolog comme outil de développement plutôt que OWL pour sa simplicité de déploiement, sa vitesse de démonstration, sa librairie de fonctionnalités et pour l'hypothèse du monde fermé. Cette hypothèse permet de garantir qu'une requête est vraie, car démontrée ce qui est important dans le cas d'une validation de systèmes.

Notez que plusieurs travaux proposent de combiner les logiques descriptives et la programmation logique pour profiter de la forte expressivité des ontologies et des capacités de raisonnement de Prolog . [3; 30; 109; 116]

3.2 Paradigmes logiciels et production de systèmes

La section précédente a présenté des approches pour modéliser et raisonner sur les connaissances d'un domaine. C'est une de ces approches qui est employée dans ces travaux pour vérifier qu'une procédure de mesure est conforme aux règles de la métrologie.

La procédure de mesure étant validée, il faut pouvoir créer des systèmes logiciels capables d'assister l'utilisateur lors de ses activités de mesure. Cette section présente trois paradigmes permettant l'implantation de systèmes logiciels : les paradigmes objet, composant et agent.

Les systèmes logiciels sont implantés en utilisant des langages de programmation. Ces langages ont évolué et proposent des abstractions toujours plus puissantes pour simplifier le travail des développeurs tout en augmentant la réutilisation et la maintenabilité des systèmes développés.

3.2.1 Paradigme objet

Le paradigme objet est l'approche la plus employée actuellement pour représenter et implanter les systèmes logiciels. Les approches impératives (pascal, assembleur, C) représentent un système par une suite séquentielle d'actions (fonctions) qui permettent de manipuler l'état du système (variables). Le paradigme objet propose d'ajouter de nouvelles entités nommées objets. Un objet est une structure logicielle qui encapsule un état et propose une interface permettant de le manipuler [13]. Un système logiciel se compose d'un ensemble d'objets qui communiquent entre eux. C'est l'interaction de l'ensemble des objets qui permet de réaliser le comportement global du système.

Le paradigme objet permet de diminuer la complexité du développement d'un système logiciel en le décomposant en objets aux buts et portées plus limitées. Un système complexe se décompose donc en un ensemble de sous-systèmes simples et maîtrisés. Ce qui permet de rapprocher la forme du système logiciel du système réel [36]. La figure 3.6 montre la structure d'objets employée par une application permettant d'estimer le prix d'une installation de panneaux solaires simple. Il y a des objets pour représenter les différents éléments du système observé : toit, capteur de distance, panneau solaire, prix. Ces objets communiquent pour mesurer le toit, calculer le prix de l'installation et proposer une interface de pilotage.

Les objets sont les éléments d'un système logiciel lors de son exécution. Le développeur ne décrit pas tous les objets du système. Il définit des

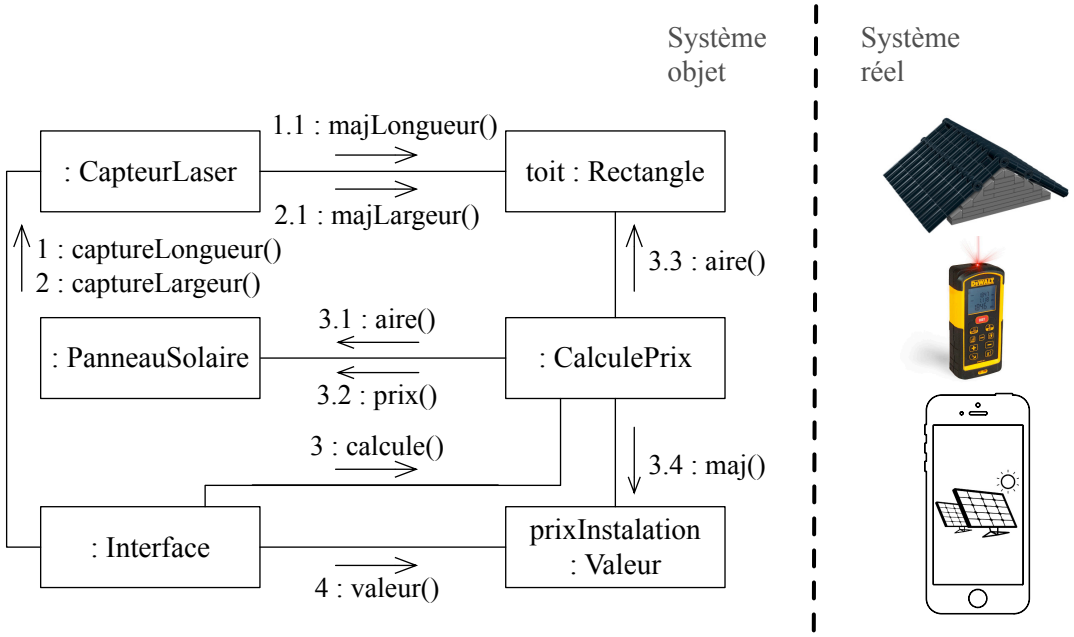

Figure 3.6 – Utilisation d'objets logiciels pour formaliser une application mobile d'estimation du prix de panneaux solaires pour un toit.

classes, des modèles décrivant chaque type d'objet. La classe définit l'ensemble des capacités d'un type d'objet et il est possible de créer (instancier) autant d'objets que nécessaire à partir d'une classe donnée. L'autre mécanisme présent est l'*héritage*. Une classe héritant d'une autre en prend les caractéristiques afin de la spécialiser. L'intérêt étant de créer des familles d'objets. Il est possible de manipuler les objets soit en tant qu'entités spécialisées soit en tant qu'entité appartenant à une famille (*généricité*).

Un objet instancié ne peut être adressé que par les fonctions et variables que le développeur a décidé de laisser accessibles. Cette *encapsulation* des mécanismes de fonctionnement est analogue à ce que proposent les systèmes du quotidien. Une voiture est un objet très complexe, mais cette complexité est cachée pour ne présenter que quelques interactions à l'utilisateur. L'encapsulation des propriétés des objets simplifie leur utilisation, leur vérification et leur réutilisation dans d'autres systèmes.

De plus, il apparaît que les programmes fondés sur les concepts objets tendent à réutiliser les mêmes structures. Ces structures ont été identifiées depuis plusieurs années et diffusées pour permettre d'appréhender correctement les problèmes qu'elles permettent de résoudre [134]. Elles sont appelées *patrons de conception*. Bien que génériques, ils peuvent présenter des spécificités en fonction du langage employé [11; 44; 54].

L'avantage principal du paradigme objet est la simplicité de développe-

ment et de compréhension du système de par la décomposition en entités simples qu'il propose. Cependant, la construction du système est fortement liée aux objets qui le composent. Il n'est possible de remplacer un objet dans une structure que par un objet qui propose au minimum la même interface. La réutilisation des objets est en réalité compliquée puisqu'ils sont souvent définis pour répondre à un besoin spécifique du problème adressé. Il faudra donc souvent mettre en place des mécanismes d'adaptations pour faire correspondre l'interface d'un objet déjà implémentée et celle nécessaire pour le système [57; 73; 94; 98].

Les paradigmes composant et agent sont des alternatives développées pour pallier à ces limitations.

3.2.2 Paradigme composant

L'approche composants est une évolution de l'approche objets qui propose de développer des solutions logicielles sous la forme d'un ensemble de systèmes autonomes capables de répondre à des messages. Un système devient donc un ensemble de sous-systèmes interagissant en partageant des données et en demandant des traitements spécifiques à des systèmes spécialisés [128].

Un composant est donc un système entier. Les composants logiciels présentent deux types d'interfaces, les interfaces fournies, qui permettent de proposer des services et les interfaces requises, qui servent à accéder à des services pour calculer les réponses fournies [43]. La métaphore correspondante est celle des composants électroniques. Il est possible d'acheter un composant avec des caractéristiques précises. Son fonctionnement interne est généralement secret, mais il fournit un service (remplit un rôle), du moment que ses conditions d'utilisation (alimentation, connexions) soient respectées.

Il est possible de créer des composants complexes à partir d'un assemblage de composants existants. Ces composants sont dits composites. L'intérêt étant de pouvoir fournir les services d'un ensemble de composants tout en simplifiant l'accès à ce service. Les interfaces présentées par ces composants agissent directement sur le composant, ou permettent d'interagir avec les composants internes [57].

Tant que ses conditions d'utilisation sont respectées, un composant peut être intégré soit au sein de la solution logicielle, soit indépendamment [21]. Cela permet de décentraliser l'exécution de tâches sur des systèmes distants ce qui permet de limiter les ressources consommées par la machine de l'utilisateur, de diminuer les accès à des bases de données distantes en gérant des requêtes complexes dans un service au niveau de la base ou

encore de proposer des services en ligne.

L'intérêt majeur de cette approche est que tout composant, ou assemblage de composants du système peut être remplacé, à tout moment, par un autre composant, assemblage de composants, s'il propose un ensemble d'interfaces identiques (à condition que l'état interne du composant ne soit pas une ressource unique et critique du système). Ainsi, les mises à jour d'un système sont simplifiées.

Pour que le couplage entre les composants soit le plus faible possible, la communication entre les composants se fait généralement en passant par des intermédiaires de communications. Ceux-ci prennent plusieurs noms en fonction des frameworks (port, connecteur, ...). En plus de prendre en charge les communications entre les composants, ils peuvent assurer leurs cohérences en imposant qu'elles n'aient lieu qu'entre des interfaces compatibles. Ils peuvent également simplifier la communication entre deux composants. Les connecteurs peuvent, communiquer des valeurs, des appels de fonctions, convertir les messages (changer le format de données) ou encore simplifier la communication entre deux composants [81].

La figure 3.7 schématise un système de composants permettant de remplir le même rôle que l'application modélisée figure 3.6.

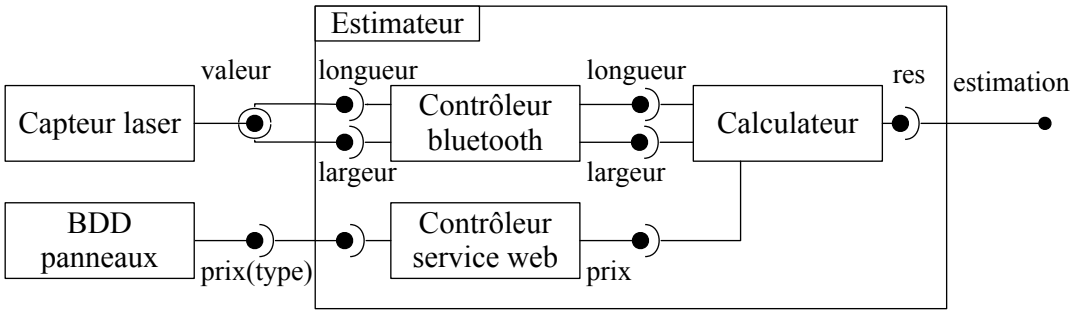

Figure 3.7 – Utilisation de composants logiciels pour formaliser une application mobile d'estimation du prix de panneaux solaires pour un toit.

L'*Estimateur* est un composant composite chargé de calculer le prix d'une installation de panneaux solaires. Il présente 3 interfaces permettant de récupérer les informations sur le monde extérieur et une interface qui fournit le résultat de l'estimation. Deux composants distants sont présents sur la figure, le *Capteur laser* qui est embarqué dans le capteur électronique et le *BDD panneaux* qui est un service web mis à disposition par le fournisseur de panneaux solaires. Les composants *contrôleurs* permettent de prendre en charge les communications avec les capteurs ou le service web.

Habituellement, les composants communiquent de façon synchrone. C'est-

à-dire qu'ils présentent un comportement passif. Ils ne sont activés que par la réception d'un message ou par l'appel d'une fonction sur leurs interfaces ; ils traitent ce message et renvoient le résultat pour ensuite attendre le suivant. Ce qui rend leur utilisation délicate dans des contextes d'exécutions distribuées ou temps réel [94]. Malgré tout, certains travaux proposent l'emploi de connecteurs capables de stocker les messages, pour apporter une exécution asynchrone au système [94; 105].

3.2.3 Paradigme agent

Le paradigme agent est proche de celui des composants. La différence majeure étant que les agents sont des entités élémentaires, qui sont soumises à une évolution asynchrone et surtout sont coordonnées en sociétés. Ils sont généralement employés dans des systèmes impliquant des exécutions concurrentes [87].

Un agent évolue indépendamment des autres agents du système. L'intérêt des agents est de simplifier la gestion et la réactivité d'un système. Ils assurent l'exécution en parallèle de toutes les tâches du système. Ce qui permet la création d'applications réactives tout en masquant les spécificités des mécanismes d'implantation [26; 69; 98; 124].

Un agent est dédié à la prise en charge d'une tâche spécifique. Pour réaliser cette tâche, il peut observer l'état de l'environnement dans lequel il évolue pour adapter son état interne. En fonction de ses observations, il sélectionne l'action à réaliser. Enfin, l'agent réalise l'action qui peut mettre à jour son état interne, son environnement pour les agents à effets de bord (blackboard [48]) ou communiquer avec les autres agents [98]. Les observations et modifications qu'un agent peut réaliser sur son environnement sont limitées à ses besoins et passent par des interfaces, comme pour les composants.

Le comportement global d'un système à agent est issu d'un effet dit d'émergence [93]. En effet, chaque agent dédié à la réalisation de tâches simples. C'est l'interaction des différents agents de la société qui permettra de faire émerger le comportement global du système.

Placer des agents au sein d'un système n'est pas suffisant pour que leurs comportements puissent réaliser des tâches complexes. Il faut organiser leurs interactions. Plusieurs types d'organisations existent comme les hiérarchies, les équipes ou les coalitions [33]. La première impose un agent qui contrôle les agents sous ses ordres ; dans la seconde tous les agents travaillent pour maximiser les chances d'atteindre un but unique commun et dans la dernière, les agents s'allient pour optimiser leurs buts propres.

La communication entre les agents n'est normalement réalisée que par

messages. Contrairement aux composants, étant autonomes, ils ne sont pas censés pouvoir répondre à des appels de fonctions qui modifieraient leurs flux d'activités. Les messages sont passés au travers d'interfaces qui permettent à l'agent d'observer ou d'éditer l'environnement dans lequel il évolue. Il est à noter que si l'agent peut modifier son environnement, dans une société multi-agents, il faudra réguler l'accès aux espaces partagés par plusieurs agents.

L'avantage des agents est leur capacité à construire des systèmes concurrents complexes de par leur autonomie et l'effet d'émergence. Cela implique aussi un désavantage. Les agents sont des entités souvent trop autonomes pour prédire le comportement exact du système [66]. La priorité et l'ordre de traitement des taches, le temps alloué à l'évolution de chaque agent, l'accès aux données sont autant de paramètres qui rendent le comportement du système indécidable. Plus le nombre d'agents augmente, plus le comportement est difficile à prédire.

Malgré tout, les sociétés d'agents et leurs contraintes d'exécutions permettent de maîtriser la portée du comportement du système. Ce qui permet de garantir son évolution globale. Par exemple, la structure hiérarchique est souvent proposée pour implanter le patron logiciel PAC (présentation, abstraction, contrôle) [69]. Elle implique une évolution en étages du système avec une entité Contrôle par agent PAC qui contrôle l'évolution des agents PAC de l'étage inférieur auxquels elle a accès. Le framework PI [98] propose un système à agents dont l'évolution est contrôlée. Il garantit que chaque agent actif évoluera au moins une fois dans un cycle d'horloge du système, cycle dont la durée est paramétrée.

Les paradigmes composant et agent permettent la création de systèmes logiciels flexibles et évolutifs. Ces systèmes peuvent être assemblés au moment de l'exécution puisque les liaisons entre les entités sont plus flexibles qu'avec des paradigmes objets. Chacun de ces paradigmes possède des avantages et des inconvénients. Plusieurs travaux tentent de marier ces deux approches pour en tirer parti [43; 57; 94; 98].

3.3 Modélisation et représentations de systèmes

Nous avons vu différentes approches pour implémenter un système logiciel. Un processus de développement logiciel implique une phase de description avant la conception. La description d'un système permet de définir l'ensemble des caractéristiques auxquelles il devra correspondre. Elle simplifie la communication entre les différentes parties d'un projet, le suivi de sa production, la cohérence des choix techniques ainsi que l'évolution du

système une fois produit.

La description d'un système logiciel est généralement appelée modèle. Un modèle est une représentation abstraite d'un système ou d'un phénomène. Un modèle peut présenter un système existant, pour en simplifier la compréhension (description), ou spécifier un système à concevoir (prescription). Dans les deux cas, il se trouve à un niveau d'abstraction plus élevé que le système. L'abstraction simplifie la compréhension du système en présentant un point de vue spécifique sur le modèle.

Modéliser un système à développer permet de s'abstraire des contraintes liées à la conception système [17]. En effet, plutôt que de s'intéresser aux détails d'implantation (code, contraintes système) le modèle peut synthétiser la structure globale, les relations entre les entités ou encore l'ordonnancement des communications entre les entités du système. Un modèle de prescription se place donc au même niveau que le plan d'un bâtiment. Il peut être employé pour implémenter, vérifier ou expliquer le système modélisé (cf section 3.4).

Les figures 3.6 et 3.7 sont des modèles de systèmes logiciels construits respectivement à l'aide d'objets ou de composants. Ces diagrammes masquent le code permettant d'implanter le système pour en faire une description structurelle simple à communiquer.

Un modèle est décrit en utilisant un *langage de modélisation* [55]. Un langage est un ensemble de termes qui sont liés par des règles syntaxiques et sémantiques. La syntaxe d'un langage décrit l'ensemble des phrases correctes qui peuvent être rédigées à partir d'un ensemble de termes de base. La sémantique est l'interprétation d'une expression dans un domaine sémantique donné.

Le code 3.1 décrit un langage de base nommé S+. S+ permet de décrire des expressions qui représentent l'assignation d'une valeur ou du résultat de la somme de valeur à une variable.

Code 3.1 Description de la syntaxe d'un langage simple (S+).

```
<expr>   ::=   <somme> '=' <var>
<somme>  ::=   <nbr> [+ <nbr>]*
<var>    ::=   "X"
<nbr>    ::=   "0" | "1" | "2"
```

Un langage définit l'ensemble de tous les modèles qui peuvent être décrits en l'employant. S+ permet de décrire l'ensemble de toutes les expressions de la forme $\sum_{i=1}^{n} V_i = Res$ avec V_i une valeur entre 0 et 2 et Res la variable x.

L'expression 1+1=X (e) est donc syntaxiquement correcte pour S+. C'est la sémantique du langage qui définit ensuite le sens associé aux expressions autorisées par la syntaxe [56]. Ainsi, si le domaine sémantique implique que x est un entier, alors le résultat de la somme de (e) qui est assigné à X est 2 (1+1). Par contre, si le domaine sémantique implique que x est une valeur booléenne, alors le résultat de la somme de (e) assigné à X est 1. En effet, pour des booléens, la somme de (e) est équivalente à '*true or true*' ;

Le langage S+ permet également de modéliser un système logiciel construit à partir de deux entités : des valeurs et des opérations. Par exemple la figure 3.8 représente un modèle du système logiciel correspondant à l'expression 0+1+2=X. Ce système est construit à partir de ports contenant des valeurs ainsi que des agents qui exécutent les opérations.

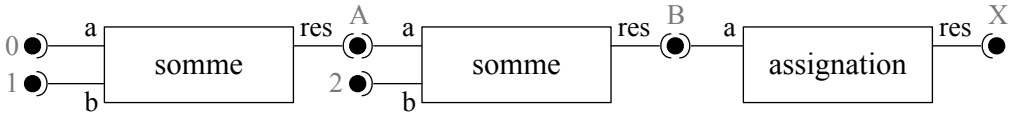

Figure 3.8 – Conception logicielle d'un système d'agents correspondant à l'expression 0+1+2=X.

Notez qu'ici chaque opération est un agent spécifique. Une autre structure possible serait d'utiliser des agents *Opérateur* avec un port d'entrée de plus pour définir l'opération à effectuer.

L'expression 0+1+2=X est une abstraction de la figure 3.8, qui est un modèle de conception (représente l'architecture du système). C'est donc un modèle du système à agents. Le modèle exprimé par le langage S+ est plus abstrait que le modèle de conception. En effet, S+ ne demande pas à s'attarder sur des détails comme les ports de connexions. Notez que le modèle de conception, bien que plus proche du niveau d'abstraction du code ne lui est pas fidèle non plus. Il ne représente pas du code, et ne s'attarde pas sur l'implémentation des comportements des agents ou des ports.

La syntaxe d'un langage peut prendre n'importe quelle forme (e.g. diagrammes, textes) [56]. La forme adéquate dépendra du système à modéliser et des affinités de la personne qui décrira le modèle. Les modèles graphiques sont souvent utilisés pour modéliser des architectures. Les modèles textuels sont concis et généralement plus pratiques pour la modélisation d'algorithmes. L'expression textuelle 0+1+2=X et la figure 3.8 illustre bien ces propos. L'expression est concise et claire et la figure montre une vue d'ensemble du système avec dix entités et 9 liens de communications identifiés d'un seul coup d'oeil.

Les langages de modélisation sont séparés en deux catégories ; les langages génériques, ou *General Purpose Languages* (GPL) et les DSLs [45].

3.3.1 Langages génériques et UML

Les GPL sont des langages spécifiés pour décrire des systèmes appartenant à des domaines divers. Ils proposent des capacités de description larges avec un niveau d'expressivité suffisant pour permettre la description de toutes les entités qui peuvent se trouver dans un système. Le langage Unified Modeling Language (UML) entre dans cette catégorie.

UML [106] est un langage graphique développé dans les années 90 pour unifier les différents langages utilisés jusqu'alors pour décrire un système logiciel. Ces langages étaient dédiés à la description d'une des propriétés du modèle (e.g. architecture, comportement). UML permet de modéliser tous systèmes logiciels. Il peut décrire la structure du système, les interactions entre les entités qui composent le système et le comportement du système ou de ses entités.

Il a été conçu dans les années 90, pour répondre au manque de formalismes pour la modélisation de systèmes logiciels, spécifiquement pour ceux construits avec le paradigme objet. Il est rapidement devenu une référence tellement répandue qu'il est délicat de citer un autre GPL [1].

La suite UML propose un ensemble de 14 types de diagrammes différents qui permettent de modéliser les différents aspects qui entrent en jeu dans le développement d'un système. Les diagrammes employés dans ces travaux sont les diagrammes d'objets (figure 3.6), les diagrammes de composants (figure 3.7) et les diagrammes de séquences.

UML est associé à une sémantique semi-formelle. En effet, le langage est décrit par une syntaxe formelle ; mais étant dédié à la description de tout système, il est associé à une sémantique modulaire. Elle est suffisamment souple pour permettre à UML de représenter tout système. Ce qui est une des raisons de son fort taux d'adoption au sein de la communauté des développeurs. Mais c'est également une faiblesse ; un modèle UML étant sujet à interprétation, il n'est pas possible de vérifier la cohérence de tous les modèles exprimables par UML . En particulier ce sont les diagrammes d'interaction ou de séquence qui posent problème. Ainsi que l'interpretation des relations entre les différents diagrammes [7; 107]. Cette sémantique faible est un problème pour la description stricte de systèmes. Cependant c'est également une force du langage. Les contraintes d'utilisation du lan-

1. La version la plus récente (2015) peut être trouvée sur le site de l'OMG http ://www.omg.org/spec/UML/

gage étant faibles, il est plus accessible lors de son apprentissage qui est malgré tout complexe [7]. Charge aux différentes parties intervenant sur le développement de s'accorder sur une façon d'interpréter les diagrammes. Cette interprétation est souvent apportée par les outils employés lors du processus de développement.

Étant un standard omniprésent sur le marché du développement logiciel, l'utilisation d'UML s'est vue assistée par des outils. Des assistants permettent de créer et partager des modèles UML de manière graphique (Entreprise Architect [61], ArgoUml [102], MoDisco [19]) ou textuelle (plantUML [103], TextUML [80]). Certains permettent la création de modèles correspondant au code d'un système [123]. Et certains ajoutent une sémantique plus restrictive à UML pour permettre la génération de code depuis le modèle correspondant (Entreprise Architect, MoDisco).

UML est donc un langage très expressif qui est employé dans beaucoup de développements logiciels. Il permet de simplifier la communication avec les personnes non experts logiciel en leur proposant des points de vue adaptés sur le système. Cependant, la généricité du langage le rend complexe à maîtriser totalement et il faudra souvent spécifier sa sémantique pour le contexte du projet.

3.3.2 Langages Spécifiques de Domaines

Les DSL sont dédiés à la description d'un domaine spécifique [45]. Cela leurs permet de proposer un vocabulaire proche de celui employé dans ce domaine, ce qui simplifie la communication avec les personnes du métier. Dans le contexte d'un développement ancré dans un domaine spécifique, une communication accrue simplifie la compréhension du problème par les développeurs et simplifie la vérification du modèle produit par les experts.

Notez que la classification d'un langage en tant que DSL est sujette à interprétation. En effet, UML pourrait être vu comme une suite de langages dédiés à la modélisation de systèmes logiciels impliquant des paradigmes objets ou composants. Néanmoins, l'intérêt et la force d'un DSL sont de proposer un vocabulaire le plus restreint et le plus ciblé possible pour le domaine. En effet, plus l'ensemble des expressions est réduit et proche du domaine plus il est simple à maîtriser.

Le langage S+ présenté précédemment (cf. code 3.1) associé à une sémantique d'algèbre mathématique est un DSL pour la modélisation d'additions. Sa syntaxe propose des termes du domaine (+, =, valeur, variable) et contraint à les assembler correctement. Toute personne avec des bases en mathématiques est capable de modéliser des opérations avec ce langage. La modélisation proposée par la figure 3.8 est déjà moins aisée à prendre

en main. Elle requiert de comprendre ce que sont les ronds, les boîtes, et les connecteurs, et ce, parce que le langage est plus proche du domaine du développement que de l'algèbre mathématique.

Les outils pour créer des DSL sont variés. Il est possible d'employer des descriptions textuelles de la grammaire (BNF [68]) et des générateurs de compilateurs (AntLR [92]). Il est également possible de restreindre les modèles UML en ajoutant une nouvelle sémantique à l'aide de contraintes (OCL [130]).

Les représentations des connaissances peuvent également servir à la création de langages. Les ontologies sont typiquement dédiées à la modélisation des connaissances d'un domaine. Elles permettent aisément d'allier syntaxe et sémantique avec les liens sémiotiques qu'elles permettent de décrire entre les entités [126; 127]. Prolog peut être employé pour l'analyse syntaxique de textes [27]. La sémantique peut également être analysée si elle est encodée sous forme de règles et que le texte peut être transformé en un ensemble de clauses.

Des outils plus évolués pour la création de langages impliquent des approches de métamodélisation (cf section 3.4). Ces outils proposent des langages associés à des sémantiques formelles capables de modéliser des DSL pour les proposer ensuite au travers d'éditeurs dédiés.

Il est à noter que, plusieurs approches permettent la définition de la syntaxe de langage de modélisation (cf. section 3.3) mais celles qui permettent d'en définir la sémantique sont rares [56].

Nos travaux visent à proposer des solutions pour la création d'applications impliquant des procédures de mesures. L'utilisation d'un DSL dans ce contexte permet de modéliser des systèmes en employant un vocabulaire proche de celui de la métrologie. Cela permet un cycle de développement simplifié puisque les clients pourront décrire leurs systèmes avec les termes de la métrologie qu'ils emploient quotidiennement et l'architecte logiciel n'aura qu'à employer des termes proches pour modéliser la solution correspondante.

3.4 Ingénierie Dirigée par les Modèles

Les trois sections précédentes ont décrit des approches pour représenter des connaissances, pour concevoir des systèmes logiciels et pour les modéliser.

Une KR d'un domaine produit une solution capable de répondre à des problèmes spécifiques à ces connaissances. La réponse est inférée par un raisonnement sur l'ensemble des connaissances.

Les paradigmes de conception logiciels sont spécialisés dans la réalisation de logiciels aux portées plus ou moins larges. La conception implique de prendre en compte les contraintes spécifiques à l'implantation du système dans une plateforme logicielle. Ces paradigmes permettent de répondre à ces contraintes en proposant de réutiliser des structures qui y répondent.

La modélisation d'un système permet de décrire et partager les entités et structure le composant. De plus il est possible de spécialiser la description du système en fonction du point de vue (client, spécialiste du domaine, codeur) et du but du modèle (structure, comportement, communication).

Ces trois méthodes (KR, paradigmes de conceptions et modélisation) sont des solutions génériques qui permettent la représentation, le raisonnement et l'implantation de tous systèmes logiciels. Leur combinaison répond aux besoins d'un cycle de développement complet comprenant la description d'un système, sa vérification et son implantation. Cependant, de par leurs natures très différentes, ces approches imposent des points de vue, des syntaxes, des fonctionnements et des buts spécifiques et distincts (figure 3.9). Ces divergences de contextes d'utilisations font qu'elles ne peuvent pas cohabiter telles quelles.

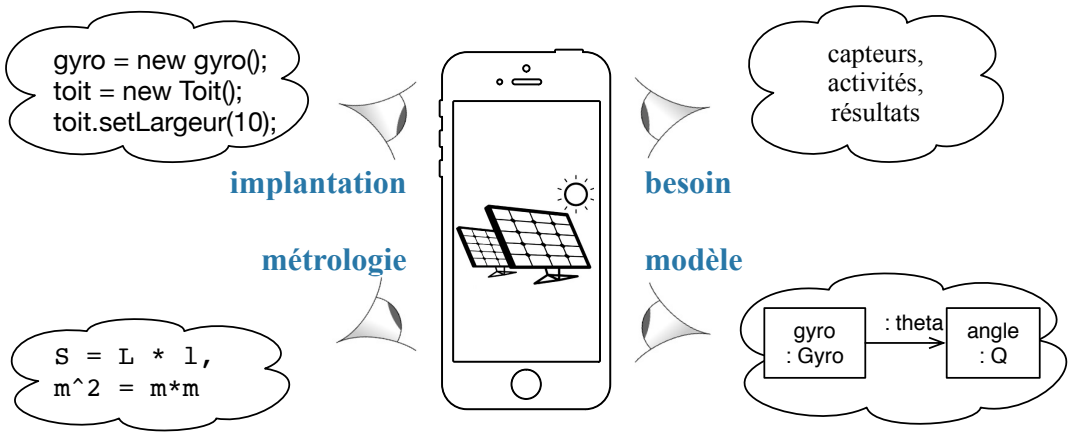

Figure 3.9 – Prisme imposé par les différents points de vue et leurs contraintes sur un système.

L'*Ingénierie Dirigée par les Modèles* (IDM), ou Model Driven Engineering est un paradigme de développement qui propose des approches permettant de gérer ce prisme par lequel passe un système en fonction des besoins à chaque étape du cycle de vie d'un logiciel. En faisant des modèles des entités de premier ordre [17; 115], l'IDM permet de modéliser les systèmes et leurs comportements à un niveau d'abstraction plus élevé. Cette relation se retrouve, par exemple, entre les bâtiments et leurs plans dans

le génie civil. Cela simplifie le développement de systèmes en proposant de focaliser la réflexion sur le but du système modélisé plutôt que sur les contraintes liées à l'implantation. Un système sera donc décrit par un ensemble de modèles. Chaque modèle représente un point de vue spécifique sur le système étudié, qui expose les points d'intérêt du système pour les personnes ou outils qui l'emploieront.

3.4.1 Modélisation et métamodélisation

L'idée de l'IDM est de considérer que tout système peut être modélisé. En considérant un langage de modélisation comme un système, il est également possible de le modéliser.

Cette considération qui a mené à la création d'une architecture récursive qui permet de modéliser un système avec un langage, qui lui-même peut être modélisé par un langage. Cette architecture récursive est présentée par la figure 3.10. Elle est répétée jusqu'à ce qu'un métamodèle soit auto représentatif, impliquant qu'il représente un langage capable de le modéliser.

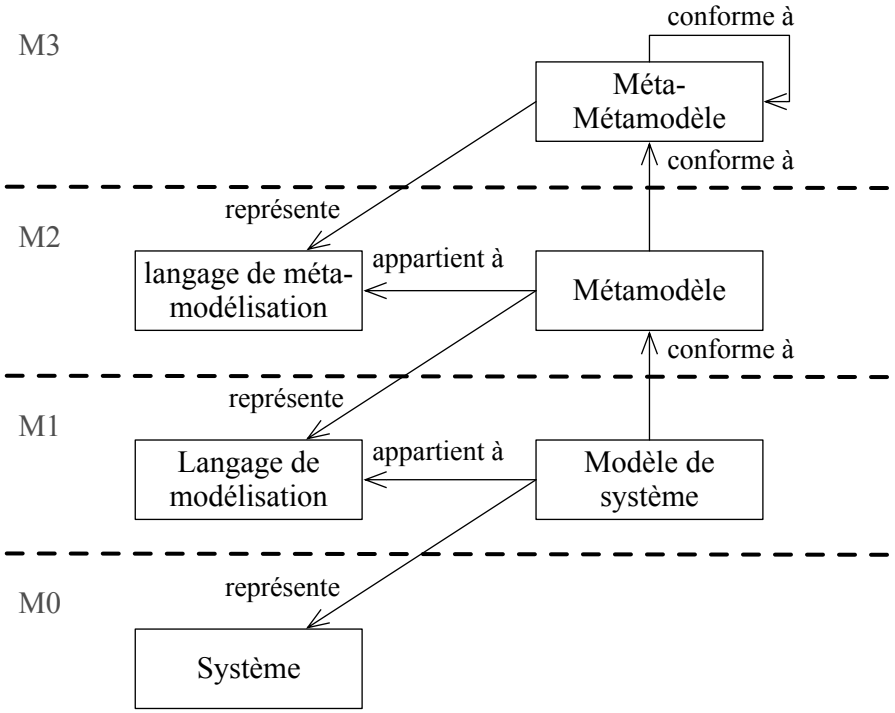

Figure 3.10 – Architecture récursive de métamodélisation proposée par la MDA pour modéliser les systèmes et les langages de modélisation.

laarchitecture dirigée par les modèles, ou Model Driven Architecture

(MDA) est un framework conceptuel de référence pour l'approche IDM proposé par l'Object Management Group (OMG) [91]. La MDA est une approche dédiée au développement de système qui propose trois niveaux de modélisation dont le niveau d'abstraction est de plus en plus proche de la solution :

— un modèle représentant le contexte, les besoins et le but de la solution,

— un modèle qui décrit le comportement et la structure de l'application,

— un modèle qui contient l'ensemble des informations concernant le comportement et l'execution de l'application sur une plateforme spécifique.

Pour décrire ces modèles, l'OMG propose une hiérarchie de métamodélisation à quatre niveaux, identifiés sur la figure 3.10 :

— M0 : implémentation du système ou instance du modèle,

— M1 : modèle du système,

— M2 : métamodèle définissant le langage de modélisation,

— M3 : métamétamodèle auto-représentatif.

L'OMG propose le *Meta-Object Facility* (MOF) [91], un métamétamodèle qui permet de décrire UML au niveau M2. D'autres architectures de métamodélisation existent comme l'*Eclipse Modelling Framework* (EMF) [117] qui emploie le métamétamodèle Ecore. Ou encore le *Diagrammatic Predicate Framework* (DPF) [72] qui lui propose une architecture récursive avec un nombre potentiellement infini de niveaux. Ce qui augmente les capacités de séparation entre les concepts exprimés à chaque niveau de métamodélisation et donc dans chaque langage. La figure 3.11 montre l'architecture de métamodélisation correspondant au langage S+ défini à la section 3.3. Le formalisme employé pour décrire ce modèle est celui proposé par DPF.

Conformément à DPF, le métamodèle DPF est employé au niveau M4. IlM4) est utilisé pour décrire un langage de modélisation LM4. LM4 permet de décrire le modèle M3. Ainsi il y a une relation entre chaque élément de M4 et l'un de ceux disponibles sur M3. Ces relations apparaissent sur le diagramme comme *:Node* ou *:arrow*. Le même principe est appliqué pour les (méta)modèles des niveaux inférieurs, qui sont décrits ultérieurement. Pour tous les liens qui ne présentent pas de multiplicité ; c'est la multiplicité "$*$" qui est sous entendue. Un lien avec plusieurs noms sous-entend l'existence de plusieurs liens identiques. Ils ne sont pas affichés par souci de clarté du diagramme.

Le langage décrit par le métamodèle M2 est S+. Le modèle de la figure 3.8 peut être décrit à l'aide de ce langage. La seule différence sera que les ports de connexion seront des instances de *Variable* ou de *Valeur*.

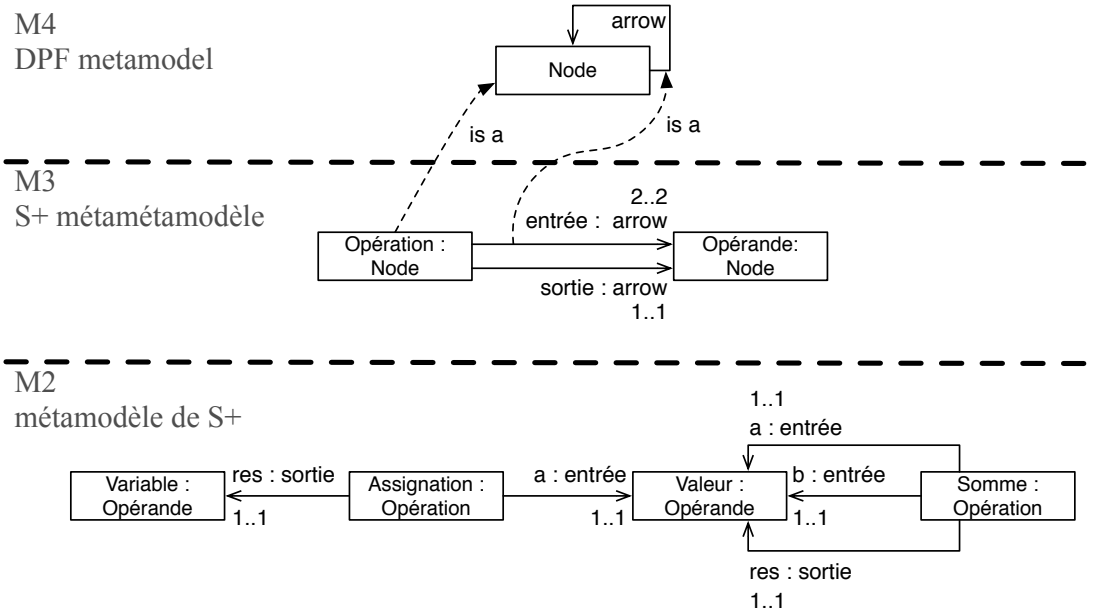

Figure 3.11 – Architecture de métamodélisation du langage S+ décrite à l'aide du formalisme proposé par DPF.

L'EMF est un framework de métamodélisation proposé par l'environnement de développement Eclipse. Il propose Ecore, un métamétamodèle, qui implémente le MOF et permet de modéliser le métamodèle d'un langage de modélisation. La suite d'outils proposés par le framework permet l'édition de ce métamodèle et la création d'un éditeur de modèles correspondant à ce métamodèle. L'EMF propose également des outils pour gérer la transformation de modèles (cf. section 3.4.2) avec pour sources des modèles issus d'Ecore. L'intégration du framework dans Eclipse implique que les outils orientent fortement vers une implantation logicielle en Java. EMF est une référence dans le domaine de la métamodélisation et l'écosystème outillé qu'il propose est conséquent.

Les outils pour ajouter de la sémantique aux langages modélisés sont des plugins à ajouter au framework et emploient l'OCL [130]. OCL est un langage de contraintes qui permet de définir sous forme de texte des contraintes sur les entités constituant un métamodèle. Lorsqu'un modèle conforme au métamodèle est créé, sa conformité aux contraintes OCL peut être validée, ce qui permet d'effectuer une validation sémantique du modèle ou simplement de restreindre l'expressivité du langage créé. Le langage et ses contraintes se retrouvent donc définis à l'aide de deux langages différents et sont stockés dans deux fichiers séparés. Cela implique que lors de l'évolution du langage il faudra également penser à faire évoluer les

contraintes.

Le DPF est également un framework de métamodélisation. Il est constitué de deux outils, le DPF workbench pour développer des DSL et des éditeurs de modèles et un plugin XPAND pour créer des transformations de modèles issus du framework. Ce framework propose une architecture de métamodélisation sans limites dans le nombre de niveaux. Le métamodèle DPF est le langage auto représentatif qui est utilisé comme sommet de l'architecture. Ensuite l'utilisateur peut modéliser autant de niveaux que nécessaire. Cela permet d'encore plus fortement séparer les notions d'implantations et de modélisations [72]. Pour cela, le DPF workbench propose un éditeur de modèles graphiques qui propose un langage correspondant au métamodèle demandé.

Le DPF emploie les transformations de graphes et la théorie des catégories pour générer les différents éditeurs et assurer de la cohérence entre le langage et le modèle créé. Chaque modèle est représenté par une spécification qui comprend un graphe associé à un ensemble de contraintes. Une contrainte est définie par un prédicat associé à un sous graphe. L'avantage de DPF est que le modèle est décrit sous forme de diagrammes qui contiennent et le modèle et les contraintes du modèle. Par exemple, la figure 3.11 montre un type de contraintes de DPF. Les multiplicités indiquent le nombre d'instances de la relation qui peuvent exister sur un modèle. Lors de la création d'un modèle, l'éditeur vérifie que le modèle correspond au métamodèle (avec ses contraintes) en vérifiant que tout élément du modèle présente une relation homomorphique vers le métamodèle.

La transformation de modèles issus des éditeurs du DPF workbench est possible à l'aide d'un plugin utilisant le métalangage XPAND agrémenté de méthodes d'accès aux différents éléments du métamodèle (instance, relation, contraintes) [6; 125].

Bien que DPF soit beaucoup moins reconnu que EMF, il présente l'avantage de proposer un environnement épuré, simple à prendre en main. Le fait que les contraintes et la syntaxe d'un langage soient définies dans un seul et même diagramme simplifie les évolutions potentielles des langages. De plus, tout élément non conforme au métamodèle est immédiatement identifié au niveau de l'éditeur, ce qui garantit la création de modèles conformes et simplifie l'utilisation de l'éditeur par des utilisateurs non habitués aux DSL ou à la modélisation. Ces prises en main et possibilités d'évolutions facilitées sont les arguments qui ont été décisifs pour l'utilisation de cet outil dans ces travaux. De plus, il a été employé avec succès dans plusieurs projets impliquant la modélisation de processus [95; 108; 129].

3.4.2 Transformations de modèles

Les modèles jouent un rôle descriptif, ils servent à représenter les éléments significatifs d'un système tout en omettant les autres (cf. section 3.3). L'IDM propose d'employer les modèles pour automatiser la création de systèmes logiciels. Cette automatisation est réalisée à l'aide des *transformations de modèles* [82; 114].

Une transformation de modèle est une entité qui, à partir d'un modèle d'entrée E, génère un modèle de sortie S. La transformation sera notée $T_{E \to S}$. Cette génération peut être vue comme une projection des éléments de E vers ceux de S . $T_{E \to S}$ respecte une définition $D_{M(E) \to M(S)}$ qui se trouve au niveau des métamodèles de E et g respectivement M(E) et M(S) (cf. figure 3.12.

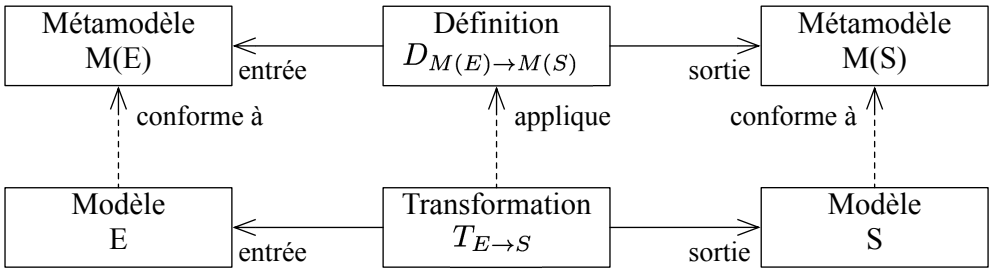

Figure 3.12 – Modélisation d'une transformation de modèle entre deux modèles et de sa définition.

Le fait que $D_{M(E) \to M(S)}$ soit exprimé au niveau méta permet de la rendre applicable à l'ensemble des modèles conformes au métamodèle puisqu'un langage définit l'ensemble des modèles qu'il peut exprimer. Ainsi $Tn_{En \to Sn}$ n'est qu'une instance des multiples transformations résultantes de $D_{M(E) \to M(S)}$.

Il existe différents types de transformations de modèles[17; 31; 82], catégorisés en fonction de leurs schémas d'exécutions, de degrés d'automatisation, de définitions ou encore de buts. Les trois premières catégories ne sont que des détails d'implantation. La catégorisation en buts sépare les transformations en deux groupes, les transformations verticales ou horizontales. Les transformations verticales impliquent que M(E) soit à un niveau d'abstraction supérieur à M(S). Elles servent particulièrement à projeter les éléments constituant un modèle vers un modèle contenant des détails de l'implantation du système, voire vers le code qui implante le système [47]. Ces transformations sont généralement des transformations modèle vers texte, ou *Model to Text* (M2T). Les transformations horizontales impliquent que les métamodèles M(E) et M(S) soient au même niveau d'abstraction. Elles permettent de changer le langage (typiquement la syntaxe) dans le-

quel est exprimé le modèle ou d'automatiser les mises à jour du modèle [83]. Ces transformations sont généralement des transformations modèle vers model, ou *Model to Model* (M2M). Il existe cependant des transformations verticales M2M (e.g. apport des spécificité d'une plateforme à un modèle indépendant de toute plateforme) ou des transformations horizontales M2T (e.g. changement de syntaxe concrète).

Notez que rien n'empêche M(S) et M(E) d'être le même métamodèle. Dans ce cas, la projection ne changerait pas l'ensemble dans lequel se trouvent les éléments de E. Une telle transformation peut avoir pour but de réorganiser les éléments pour simplifier leurs lectures. Elles sont également utilisées pour la complétion automatique de modèles [96].

Le code 3.2 décrit le processus de la transformation du modèle figure 3.8 vers une implémentation sous forme d'agents logiciels.

Code 3.2 Processus de transformation entre les modèles décrits avec S+ et le code d'implémentation correspondant.

```
soit ports une liste de vecteurs <Nom, Port>
soit agents une liste de vecteurs <Nom, Agent>

pour chaque instance de noeud n du modèle
        créer un objet o de type n.type()
        ajouter <n.nom(),o> dans la liste correspondant au type n.
           type()

pour chaque instance de relation r du modèle
        agent = agents.get(r.source().nom())
        port = ports.get(r.destination().nom())
        agent.connect(r.type(),port)
```

L'IDM permet donc, avec la métamodélisation, de modéliser des DSL et des systèmes logiciels à des niveaux d'abstractions suffisamment élevés pour ne pas considérer les détails liés à l'implantation. Ces détails sont ajoutés au modèle par des transformations de modèles automatisées. Les transformations sont spécifiées en décrivant les définitions des projections entre les entités du métamodèle décrivant le modèle du système et celui du code d'implantation ciblé. Un compilateur de code peut être vu comme une transformation verticale d'un modèle (code informatique) correspondant à un métamodèle (grammaire du langage de programmation) vers un ensemble d'instructions compréhensible par la machine, correspondant à des instances spécifiques de l'ensemble des instructions.

Rappelons que la section 3.3 décrivait que la sémantique d'un langage

est délicate à définir lors de la création d'un DSL. L'IDM permet d'adjoindre à la syntaxe d'un langage une sémantique forte au travers des transformations de modèles. En effet, par manque de sémantique, c'est l'utilisation du modèle qui définira la sémantique qui lui est associée. Dans ce contexte, les transformations de modèles peuvent être vues comme des médias qui associent de la sémantique aux modèles [17].

Plusieurs outils permettent le développement de M2T automatiques [24; 67; 104]. Ils utilisent un métalangage qui permet de décrire la définition de la projection sous forme de templates. Le DPF workbench propose un plugin utilisant le langage XPAND. Ce plugin peut être utilisé avec pour source, n'importe quel métamodèle qui aura été défini à l'aide d'un éditeur du DPF workbench [110].

Deuxième partie

Contribution

Chapitre 4

Représentation des Connaissances de la Métrologie

« Une théorie est vraie si elle est énonçable selon les règles de la logique formelle, et si ses conséquences sont vérifiables par tout observateur. »

Jacques Attali

Sommaire

Ces travaux ont pour objectif la simplification de la création de systèmes logiciels impliquant des mesurages. Cette simplification se fait sur trois axes :

1. assurer la cohérence du mesurage avec les règles de la métrologie,
2. diminuer l'écart sémantique entre l'architecte logiciel et le client lors de la modélisation des systèmes,
3. assister la création du logiciel à partir de son modèle.

Ce chapitre se focalise sur le premier des trois points. Il propose premièrement un modèle conceptuel de la métrologie telle qu'elle a été décrite dans le chapitre 2. Ce modèle sera employé comme modèle de référence dans les trois chapitres suivants. Il permet de définir l'ensemble des termes qui constituent la syntaxe de la métrologie employée dans ces travaux.

Ensuite, une représentation des connaissances de la métrologie est proposée. Nommée Metrology Knowledge Schema (MKS) cette représentation des connaissances, ou Knowledge Representation (KR) est définie à l'aide d'un ensemble de clauses de Horns en language SWI-ProLog [133]. Elle permet le raisonnement sur l'ensemble des connaissances que sont l'analyse dimensionnelle, l'analyse des échelles de mesures et la conversion d'unités.

La présentation du MKS est séparée en quatre sections complémentaires. La première présente comment les différents termes de bases constituant les unités, les dimensions et les systèmes de grandeurs sont inférés. La seconde décrit les règles d'analyses qui permettent de réaliser l'analyse dimensionnelle et l'analyse des échelles de mesures. La troisième section étend les capacités expressives du MKS afin de proposer un langage capable de modéliser des procédure de mesures, le Measurement Knowledge Query Language (MKQL). Le MKQL propose des termes proches de ceux employés dans la métrologie. Cette section illustre également les capacités de raisonnement du MKS qui permettent la vérification de la cohérence de procédures de mesures selon les règles de la métrologie. En effet, chaque modèle décrit par se langage est une requête de vérification, par le MKS, de la procédure modélisée. La quatrième section ajoute les règles qui permettent de raisonner sur l'ensemble des conversions entre les différentes unités d'un système. Enfin, la cinquième section présente les différents mécanismes spécifiques à SWI-ProLog qui sont employés pour augmenter les capacités d'analyse du MKS, pour finalement conclure sur les capacités de mise à l'échelle de l'approche en décrivant la possibilité de créer une KR spécialisée pour les besoins d'une analyse dans un domaine spécifique ou une KR pour l'analyse de tout type de procédure de mesures.

4.1 Modèle conceptuel et approche proposée

Les termes de la métrologie présentés dans le chapitre 2 sont employés pour, premièrement, décrire les mesurages et la transposition d'évènements empiriques en grandeurs et deuxièmement pour décrire la sémantique associée à la manipulation de ces grandeurs. Cette sémantique contraint les fonctions de mesure pour refléter dans le domaine abstrait les relations empiriques du domaine physique. La combinaison de ces éléments permet d'appliquer des fonctions sur des abstractions formelles et d'en déduire des relations physiques, plutôt que de les trouver en manipulant des objets du monde physique. De plus, la sémantique de la métrologie permet d'assurer la cohérence entre les abstractions et les relations physiques qu'elles représentent.

L'utilisateur final, qui décrit la procédure de mesure, et l'architecte logiciel, qui implante la procédure dans un logiciel, possèdent rarement une expertise en métrologie. Cependant, cette expertise est très importante pour le développement de procédures correctes. L'utilisation des concepts de la métrologie dans un processus de développement permet de garantir la qualité des mesurages du système résultant.

L'application de l'analyse dimensionnelle et de l'analyse des échelles de mesures permet de vérifier la procédure décrite par l'utilisateur (cf chapitre 2), ainsi que sa correspondance avec la procédure impliquée dans le système développé.

Plusieurs propositions existent pour modéliser les procédures de mesures [58; 64; 111] pour appliquer la sémantique de la métrologie [23; 100] ou pour prendre en charge des grandeurs non-physiques [51]. Cependant, nous l'avons indiqué dans le chapitre 2, aucune de ces solutions n'est standard ; elles proposent des interprétations différentes de la métrologie avec des implantations et des encodages différents [42]. Aucune solution ne répond au besoin de l'intégration de la sémantique de la métrologie dans un processus de développement logiciel sans expertise du domaine.

Nous proposons une approche IDM pour le développement de solutions logicielles impliquant des procédures de mesure. Une telle approche impose de définir la syntaxe et la sémantique de chaque langage impliqué, autrement, il n'est pas possible de décrire des définitions de transformations.

C'est pourquoi, afin de pouvoir se focaliser sur l'ensemble des propriétés de la métrologie, qui sont importantes pour le contexte de ces travaux en assurant leurs maitrises, nous proposons un nouveau modèle conceptuel pour la description de procédure de mesure dérivé de la métrologie. Le mo-

dèle conceptuel de la métrologie, ou *Metrology Conceptual Model* (MCM) est présenté par la figure 4.1. Ce modèle est fondé sur le Vocabulaire International de Métrologie (VIM) [89] et comprend les éléments conceptuels nécessaires à l'application de la sémantique présentée dans le chapitre 2. Pour minimiser la charge graphique du modèle, les multiplicités associées aux relations sont définies par le nombre associé à chaque nom de relation. une relation au singulier implique une seule entité cible quand une relation au pluriel implique une ou des entités cibles. Les triangles noirs indiquent la navigabilité d'une relation quand il en existes plusieurs entre deux mêmes entités.

Un instrument de mesure est un outil qui permet à un utilisateur d'*encoder* une propriété physique en une grandeur. Un instrument indicateur est employé pour *décoder* la grandeur et présenter à l'utilisateur la valeur résultante, et cohérente au monde physique, qu'il espère obtenir de la procédure (cf figure 2.2). Une fonction de mesure est une *manipulation formelle* utilisée pour déduire des grandeurs mesurées, celles espérées en résultats.

Une procédure de mesure est la combinaison de plusieurs instruments de mesure, indicateurs et fonctions de mesures. Ces éléments sont liés par les grandeurs produites et consommées ; comme le seraient des agents au travers de ports partagés. Ce qui produit une chaîne d'éléments.

Une grandeur est l'association d'une valeur, d'une unité et d'une échelle de mesure de la RTM. Ces attributs permettent :

— la manipulation algorithmique des valeurs,

— l'application de l'analyse dimensionnelle,

— l'application de l'analyse des échelles de mesure,

— la gestion des unités à l'aide de conversions.

Le VIM propose d'employer le terme *référence*, associé à une valeur, pour décrire une grandeur. Les unités sont normalement une spécialisation des références associées aux grandeurs physiques définies par le SI. Cependant, nous cherchons à proposer un modèle accessible pour les non-experts. Le terme *unité* est entré dans le vocabulaire commun, quand celui de référence n'est que rarement associé aux mesures. C'est donc le terme *unité* est utilisé dans le modèle conceptuel.

Une relation de conversion peut être définie entre deux unités. Cette relation implique une unité source (*de*) et une unité de destination (*à*). L'ensemble de ces relations génère un graphe de conversions qui permet de convertir toute unité valide, vers une unité éligible, en suivant un chemin impliquant de multiples conversions.

Le MCM décrit les connaissances nécessaires pour la description de procédures de mesures spécifiques ainsi que pour l'application des méthodes

Figure 4.1 – Modèle Conceptuel de la Métrologie. Ce modèle est capable de représenter des procédures de mesure ainsi que les systèmes permettant les analyses dimensionnelles et des échelles de mesures.

d'analyse de la sémantique de la métrologie. De plus, le modèle inclut le vocabulaire employé par les futurs utilisateurs des systèmes logiciels produit avec ces travaux.

Les entités présentes dans le MCM sont représentées dans le Metrology Knowledge Schema (MKS), le langage de modélisation d'application d'assistance à la mesure, Measure Application Modelling Language (MAML) et l'application mobile de la plateforme de modélisation présentée dans ces travaux (cf figure 1.2). Le MKS est une KR qui encode en logique du premier ordre, ou First Order Logic (FOL) les capacités descriptives du modèle conceptuel, en y ajoutant les capacités de vérification des analyses dimensionnelles et des échelles de mesure. Le MAML est un langage de modélisation qui fusionne les entités du modèle conceptuel ainsi que celles provenant de la modélisation des processus et des interfaces utilisateur. L'application mobile est un système logiciel configurable qui accompagne un utilisateur dans la réalisation de ses procédures de mesure.

Le MCM peut donc être vu comme un métamodèle partiel de ces trois entités. L'intérêt de cette approche est de simplifier la coopération des outils proposés par la plateforme, d'augmenter la cohérence des travaux et de proposer un vocabulaire relativement homogène lors de l'emploi d'entités liées à la métrologie dans ces travaux.

4.2 Encodage des termes de la métrologie

Cette section et les suivantes présentent le Metrology Knowledge Schema (MKS), une représentation des connaissances de la métrologie encodée en logique du premier ordre. Le but de cette représentation des connaissances, ou Knowledge Representation (KR) étant de pouvoir raisonner sur l'espace sémantique de la métrologie et de vérifier la cohérence de procédures de mesurer avec ce raisonnement.

La KR comprend un système d'unités, une syntaxe pour exprimer les unités composées, deux analyses (dimensionnelle et échelles de mesure), des relations de conversions et un langage de requêtes. Ces constructions sont représentées sous la forme de faits et de règles d'inférence, qui constituent un métamodèle formel pour la représentation des connaissances de la métrologie en prédicats de la logique du premier ordre. La réalisation de la KR est écrite avec le langage ProLog ; ainsi, seules des clauses de Horn seront employées dans la description qui suit. L'implication de ProLog dans cette description est tenue à son minimum et il n'y sera fait allusion que dans la fin de ce chapitre pour illustrer les mécanismes d'optimisation du raisonnement, de traçage des résultats et de prescriptions du MKS.

La KR proposée est conçue pour être hautement configurable. Sa création et ses extensions futures devraient être réalisées avec l'assistance d'un expert en métrologie. Cependant, une fois configuré, le MKS est un système expert qui est spécifié pour être utilisé par des non-experts pour soutenir la cohérence de leurs procédures de mesures.

Les termes employés pour décrire la KR sont en anglais. Cela permet de proposer une description du MKS avec des notations plus courtes, non accentuées et donc qui sera directement implémentable. Les termes employés sont ceux du VIM [89] pour toutes traductions, il faut s'y référer. Le document présente le vocabulaire anglais et français.

4.2.1 Description des systèmes d'unités

Les systèmes d'unités simplifient la spécification d'unités en décrivant un ensemble réduit d'unités de bases et en décrivant les autres avec des combinaisons algébriques. Un système correctement défini permet d'employer l'analyse dimensionnelle. Le tableau 4.1 liste les termes et prédicats essentiels employés pour modéliser un système d'unités. Le tableau 4.2 illustre l'utilisation de ces constructions pour décrire une instance d'un système d'unités. Dans cet exemple tous les termes sont clos.

La construction d'un système d'unités est faite à partir des notations du SI. Le SI utilise des facteurs de réduction et d'augmentation, appelés préfixes, pour mettre à l'échelle de la propriété observée l'unité employée. Ceux-ci permettent de simplifier la notation des valeurs de grandeurs (e.g. micro pour 10^{-6}). Le SI préconise l'utilisation des alias pour référer à des unités simplement (e.g. newton pour l'unité $kg * m/s^2$). Pour le MCM, les alias sont aussi utilisés pour prendre en charge les traductions des unités.

L'une des propriétés d'un système d'unités est la réduction d'unités vers des vecteurs aux dimensions. En fonction du domaine, les unités communément employées (unités de références) peuvent varier. De plus, le but premier d'un utilisateur est de mesurer des types de grandeurs (longueur, surface, vitesse, etc). Les dimensions et l'analyse dimensionnelle permettent de s'abstraire des unités et de focaliser l'analyse sur les types de grandeurs impliqués dans une procédure de mesure. Ce sont donc les pierres angulaires de tout outil métrologique.

Variables	**Description**
u, v, p, etc	n'importe quelle variable
fonctions	**Description**
prefixed_unit(p,u)	l'unité u est préfixée par p
power(u,n)	l'unité u est élevée à la puissance n
product(u,v)	décrit le produit des unités u et
quotient(u,v)	décrit le quotient des unités u et v
Predicats	**Description**
is_number(n)	n est un nombre
is_atomic_unit(u)	une unité associée à une dimension de base
is_derived_unit(u)	une unité dérivée
is_prefix(p,r)	p est un préfix de ratio r
is_alias(a,u)	un alias accepté pour u
is_unit(u)	une unité simple ou composée
is_base_dim(d)	une dimension de base
is_derived_dim(d)	une dimension dérivée
is_reference_unit(d,u)	une unité de référence pour la dimension
is_convertible(u,c,v)	u est convertible en v à l'aide de c
is_convertible(u,v)	u est convertible en v

Tableau 4.1 – Termes et règles pour l'inférence d'unités, de dimensions et de systèmes d'unités

is_base_dim(length)	is_derived_dim(area)
is_atomic_unit(meter)	is_atomic_unit(mile)
is_derived_unit(are)	is_prefix(kilo,1000)
is_alias(m,meter)	is_prefix(centi,0.01)
is_alias(mètre,meter)	is_alias(km, prefixed_unit(kilo,m))
is_alias(kilomètre, km)	is_alias(cm, prefixed_unit(centi,m))
is_reference_unit(length, meter)	
is_reference_unit(area, power(meter,2))	
is_convertible(mile, 1.6, prefixed_unit(kilo,m))	
is_convertible(are, 100, power(meter,2))	

Tableau 4.2 – Instance d'un petit système d'unités

Tête	Corps
is_unit(u)	is_atomic_unit(u)
is_unit(a)	is_alias(a,u)
is_unit(u)	is_derived_unit(u)
is_unit(prefixed_unit(p,u))	is_prefix(p,n) ∧ is_unit(u)
is_unit(power(u,n))	is_number(n) ∧ is_unit(u)
is_unit(product(u,v))	is_unit(u) ∧ is_unit(v)
is_unit(quotient(u,v))	is_unit(u) ∧ is_unit(v)
is_unit(u)	is_convertible(u,c,v)
is_unit(u)	is_convertible(v,c,u)
is_convertible(u,v)	is_convertible(u,c,v)
is_convertible(u,v)	is_convertible(v,c,u)
is_convertible(u,v)	is_convertible(u,c,z) ∧ is_convertible(z,v)

Tableau 4.3 – Règles d'inférences pour la description d'unitées

4.2.2 Description des unités composées et schéma de conversions

Le tableau 4.3 liste les règles d'inférences utilisées pour construire des unités composées. Le SI considère qu'une unité peut être une unité atomique, un alias, une unité dérivée, une unité préfixée, une unité à laquelle on applique une puissance négative ou positive, un produit ou un quotient d'unités et toute unité convertible en une unité du SI. L'ensemble des règles du tableau 4.3 définit la grammaire qui décrit les expressions d'unités valides.

Ainsi, dans un système avec les alias : pascal, joule et newton ; le newton pourrait se représenter par les expressions :
$quotient(product(prefixed\_unit(kilo, gramme), m), power(seconde, 2))$,
$newton$, $quotient(joule, mètre)$) ou encore $product(are, pascal)$.

L'utilisation du prédicat is_convertible était montré dans le tableau 4.2. Ce prédicat définit une relation entre deux unités. Si une relation de conversion $c_{u \to v}$ d'une unité u à une unité v existe, elle implique qu'une relation de conversion réciproque $c^{-1}_{v \to u}$ de v vers u existe. De plus, les relations de conversion peuvent être composées. Cette composition crée un chemin de conversions. Ces deux propriétés génèrent virtuellement un graphe non dirigé, dont les noeuds sont des unités et les arêtes sont des conversions autorisées entre des unités de mêmes dimensions. Les règles du tableau 4.3 permettent d'inférer ces relations de conversions en ajoutant des propriétés de réciprocité et la transitivité aux prédicats is_convertible.

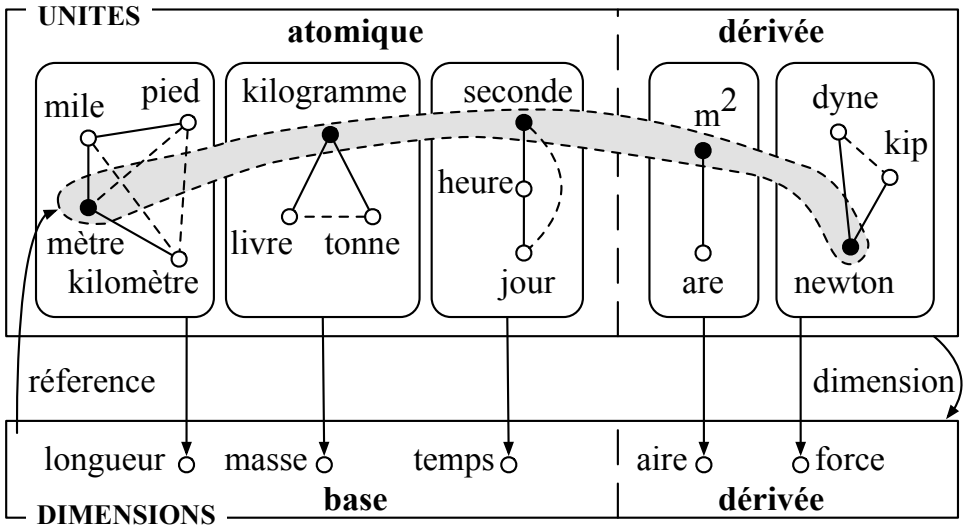

Figure 4.2 – Classification des unités par dimensions et relations de navigations par conversions. Les liens de conversion pleins présentent une relation indiquée sous la forme d'un prédicat du MKS et ceux en pointillés une relation inférée.

Ainsi, un système de grandeurs dans un MKS peut être vu comme un ensemble de dimensions et d'unités liées par des conversions ou des relations de compositions qui forment un graphe (figure 4.2).

L'ajout d'une nouvelle unité dans la KR se fait en deux temps :

1. définition de l'unité atomique ou dérivée,
2. connexion au graphe d'unités existantes ; avec un prédicat $is\_convertible$.

Toutes les unités d'une dimension, donc d'un même sous-graphe, peuvent être énumérées à l'aide de la règle $is\_convertible(u, v)$

Un système complet définit les relations entre les différentes unités dérivées et des combinaisons d'unités de base du système. Ces relations sont apportées dans le MKS par des alias (newton), des conversions, ou des définitions explicites de l'unité de référence de la dimension ; l'avantage de ces dernières étant de proposer un point d'entrée clairement défini au sous-graphe de conversion associé à la dimension.

Pour chaque dimension, une référence est choisie en construisant une section dans l'ensemble des unités classifiées (figure 4.2). Pour les dimensions dérivées, les unités de références devraient être choisies comme une combinaison d'unités de référence des dimensions de base. Cela garantit que chaque dimension de chaque unité du système peut être réduite à une combinaison de dimensions de base.

Par exemple, le tableau 4.2 montre cette définition pour les longueurs et les aires. La dimension de l'aire n'est pas décrite explicitement comme

une longueur au carrée ; son sous-graphe présente un point d'entrée défini comme l'unité m^2. Il est alors possible de remonter depuis l'unité are vers une combinaison d'unités de base qui la composent :
$are \rightarrow m^2 \rightarrow aire \rightarrow longueur^2$.

4.3 Encodage des règles d'analyses

4.3.1 Analyse dimensionnelle

Les grandeurs combinent une valeur et une unité, pour mesurer des grandeurs physiques. Les opérations sur les grandeurs manipulent des abstractions de phénomènes physiques. L'analyse dimensionnelle vérifie que ces opérations ont un sens par rapport aux règles de la physique. Puisque l'ensemble des unités employées peut être spécifique à un domaine ou à une application ; l'analyse n'est pas faite sur les unités mais sur leurs dimensions.

Règles d'analyse

Les vecteurs aux dimensions (abrégés à partir de maintenant par *d-vectors*) servent à encoder les types de grandeurs associés à toute unité. Chaque élément du vecteur est associé à une dimension de base et peut être élevé à la puissance n ; n étant un nombre entier. Par exemple, le SI propose sept grandeurs de bases, celles-ci impliquent un vecteur à sept dimensions de la forme : $< \mathrm{L}, \mathrm{M}, \mathrm{T}, \mathrm{I}, \Theta, \mathrm{N}, \mathrm{J} >$ dont les symboles représentent respectivement : longueur, masse, temps, courant électrique, température thermodynamique, quantité de matière et intensité lumineuse. Avec cet encodage, une longueur est associée au d-vector $< 1,0,0,0,0,0,0 >$, une aire au d-vector $< 2,0,0,0,0,0,0 >$ et une vitesse au d-vector $< 1,0,-1,0,0,0,0 >$.

L'analyse dimensionnelle empêche d'appliquer des opérations illégales sur des grandeurs. Pour les opérations autorisées, elle permet également de définir la dimension associée aux résultats de leurs applications.

Il est possible de concaténer deux grandeurs, si elles représentent la même propriété physique. Cette concaténation produira un résultat représentant les mêmes propriétés physiques. Les opérations d'addition et de soustraction nécessitent donc que tous les d-vectors impliqués soient identiques. Par exemple : l'opération $g_1 + g_2 + g_3 = g_{res}$, avec g_x des grandeurs n'est autorisée que si :
$d\text{-}vector(g_1) = d\text{-}vector(g_2) = d\text{-}vector(g_3) = d\text{-}vector(g_{res})$.

La combinaison de plusieurs types de grandeurs entraine la création

Constantes	**Description**
add, sub, mul, div	opérations basiques sur les quantités
Prédicats	**Description**
is_dvector(v)	v est un d-vector
is_at(d,p)	la dimension d est à la position p
is_sum(v,w,r)	r est la somme des vecteurs v, w
is_diff(v,w,r)	r est la différence de v, w
is_valid(o,v,w,r)	o est une opération valide pour les grandeurs v, w et r
is_cform(u,c)	c est la forme canonique de u
is_dvector(c,d)	d est le d-vectpr associé à la forme canonique c
is_dimension(u,d)	d est le d-vector associé à l'unité u

Tableau 4.4 – Prédicats pour inférer l'analyse dimensionnelle

d'un nouveau type de grandeur en fonction des lois de la physique. Les opérations de multiplication et de division peuvent être réalisées sur tout ensemble de grandeurs. La dimension de la grandeur résultante est respectivement définie par la somme ou la soustraction vectorielle des grandeurs combinées. Ainsi, l'opération $(g_1 * g_2)/g_3 = g_{res}$, avec g_x des grandeurs, implique :
$d\text{-}vector(g_1) + d\text{-}vector(g_2) - d\text{-}vector(g_3) = d\text{-}vector(g_{res})$.

Le tableau 4.4 montre les prédicats essentiels ajoutés au MKS pour permettre d'inférer l'analyse dimensionnelle. Ces règles permettent :

— de transformer les expressions d'unités (cf. tableau 4.3 en vecteurs aux dimensions,

— de dériver l'effet que les opérations sur des grandeurs impliquent aux dimensions.

is_dvector identifie les d-vectors valides pour un MKS donné. *is_at* identifie l'index d'une dimension de base dans le d-vector. *is_sum* et *is_diff* identifient la somme et la différence de deux d-vectors.

Le tableau 4.5 présente les faits et règles qui permettent d'appliquer les contraintes de l'analyse dimensionnelle associées aux concaténations et compositions de grandeurs. Les deux premiers faits servent à contraindre les additions et soustractions de grandeurs. Les deux règles suivantes encodent les contraintes sur les multiplications et les divisions de grandeurs.

En utilisant le prédicat *is_valid*, il est possible de requérir au MKS pour assurer la cohérence d'opérations mathématiques sur des vecteurs aux dimensions. L'unification de la requête n'est possible que si les règles de l'analyse dimensionnelle sont respectées.

Tête	Corps
is_valid(add,v,v,v)	
is_valid(sub,v,v,v)	
is_valid(mul,v,w,r)	is_sum(v,w,r)
is_valid(div,v,w,r)	is_diff(v,w,r)

Tableau 4.5 – Faits et règles pour contraindre les fonctions de mesure à respecter l'analyse dimensionnelle.

Transformation vers les vecteurs aux dimensions

La combinaison de grandeurs arbitraires génère des expressions d'unités comme $(m/(s*s))*kg$ ou $(kg*(m/s))/s$. Des expressions équivalentes, comme ces deux-ci, devraient être identifiées. Une approche pour réaliser cette identification est de transformer les expressions sous une forme canonique. Cette forme doit garantir que deux expressions équivalentes ont une forme canonique identique.

La forme canonique d'une expression d'unité est un ensemble, non ordonné, de puissances d'unités. Par exemple, la forme canonique des deux expressions précédentes est :
$\{power(meter,1)\ ,\ power(kilogram,1)\ ,\ power(second,-2)\}$.
Une forme canonique est interprétée comme le produit des éléments qu'elle contient. Elle peut être transformée en un d-vector en utilisant la règle suivante qui combine des prédicats du tableau 4.4 :
$is\_dimension(u,d) \leftarrow is\_cform(u,c) \wedge is\_dvector(c,d)$.
Le d-vecteur peut être utilisé et vérifié par les règles de l'analyse dimensionnelle à partir d'une requête au MKS.

Le prédicat *is_cform* permet la transformation d'une expression d'unité en une forme canonique. Une expression d'unité peut être vue comme un arbre syntaxique dont les opérations sont des branches et les unités des feuilles. Le processus de canonisation implique donc de visiter l'ensemble de l'arbre et d'appliquer une transformation spécifique en fonction des éléments rencontrés. La canonisation respecte donc ce schéma :

— unité de base : retourne l'unité de base à la puissance 1,

— unité préfixée : retourne la canonisation de l'unité,

— puissance d'unité : retourne la canonisation de l'unité augmentée par la puissance,

— somme, soustraction : retourne la canonisation d'une des unités,

— multiplication : retourne la somme des canonisations des unités,

— division : retourne la différence des canonisations des unités,

— alias, unité composée : retourne la canonisation de l'unité de définition.

C'est cette canonisation qui bénéficie le plus de la définition d'une unique unité de référence pour chaque dimension. En effet, cet unique point d'entrée implique également une unique canonisation, permettant de remonter vers les différentes unités de base constituant les unités composées du système.

Rappelons que la forme canonique contient au plus une occurrence des différentes unités de base du système. Ainsi, la transformation vers un d-vecteur *is_dvector* est réalisée en associant chaque unité de base à sa dimension et en assignant à chaque dimension la valeur de puissance correspondante.

Exemple

Les équations du code 4.1 représentent une procédure de mesure *P* qui implique de mesurer les longueurs et largeurs d'un toit, pour en calculer la surface, afin de définir le nombre de panneaux solaires qui peuvent y être installés.

Code 4.1 Equations de la procédure de mesure *P* qui permet de calculer le nombre de panneaux solaires à placer sur un toit.

```
largeur * hauteur = toit
toit / panneau = nbr_panneaux

largeur en mètre,
hauteur en mile,
panneau en are,
nbr_panneaux en un (sans dimmension)
```

Le code 4.2 illustre une requête d'analyse dimensionnelle de *P* au MKS. Notez que l'unité *un* est employée pour dénoter une grandeur sans dimension.

La notation employée pour la requête est celle de ProLog. Un terme qui commence par une minuscule est une constante et par une majuscule est une variable. Cette requête pourrait être également écrite avec une syntaxe sous forme de système d'agents (figure 4.3). Les d-vecteurs qui se substituent aux variables lors de l'unification de la requête sont affichés en gris près des variables correspondantes

La requête est confrontée à un MKS qui est dédié au domaine, avec un algorithme de résolution de problèmes qui prend en charge l'unification.

Code 4.2 Exemple d'une requête pour la validation de la procédure de mesure *P* par l'analyse dimensionnelle du MKS

```
?- is_dimension(mètre, D_largeur) ^
is_dimension(mile, D_hauteur) ^
is_dimension(are, D_panneau) ^
is_dimension(un, D_nbr_panneaux) ^
is_valid(mul, D_largeur, D_hauteur,  D_toit) ^
is_valid(div, D_toit, D_panneau, D_nbr_panneaux)
```

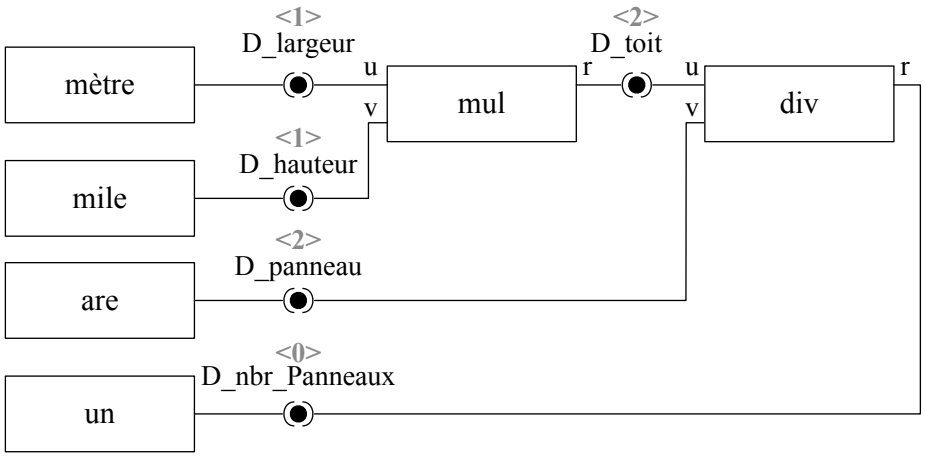

Figure 4.3 – Modèle d'agents décrivant la requête d'application de l'analyse dimensionnelle pour la procédure *P*.

Le domaine ne comportant que des longueurs, des surfaces et des grandeurs sans dimension, un d-vector contient une dimension : < L >. Les variables D_largeur et D_hauteur sont substituées par le d-vector : < 1 > ; D_panneau et D_toit par : < 2 > ; D_nbr_panneaux par < 0 >.

Ainsi *is_valid(mul, D_largeur, D_hauteur, D_toit)*
devient *is_valid(mul, <1>, <1>, <2>)*
et *is_valid(div, D_toit, D_panneau, D_nbr_panneaux)*
devient *is_valid(div, <2>, <2>, <0>)*.

Ces deux opérations respectent les règles de l'analyse dimensionnelle décrite précédemment. L'unification est un succès puisque toutes les variables ont été substituées. Alors, l'équation du code 4.1 est correcte du point de vue de l'analyse dimensionnelle et a donc du sens dans le monde physique.

4.3.2 Analyse des échelles

Ces travaux cherchent à proposer une solution qui prend en charge les grandeurs physiques, codifiées par le SI, ainsi que les grandeurs non physiques. Pour étendre la vérification des procédures de mesure, ce sont les concepts de la théorie représentationnelle de la mesure, ou Representational Theory of Measurement (RTM) qui sont utilisés.

Règles d'analyse

La RTM définit des ensembles de capacités algébriques, associées aux grandeurs, appelés échelles de mesure. Les échelles sont définies en fonction du phénomène qui est abstrait par une grandeur. Cette définition permet la manipulation de grandeurs non physiques, telles que les genres humains, les couleurs ou les notes d'examens.

Pour ces grandeurs, la définition d'une échelle est sujette à discussions. Le genre humain, comme les couleurs, présentent une échelle nominale. À moins que les couleurs ne soient définies par une longueur d'onde, dans ce cas ce sera une échelle ordinale ($\lambda_{rouge} < \lambda_{violet}$). Puisque les notes d'examens sont considérées en France comme des valeurs continues, elles sont associées à l'échelle ratio, et il sera possible de calculer la moyenne arithmétique d'un ensemble de notes. En revanche, les pays anglo-saxons emploient une notation avec des lettres. Il est impossible de calculer une moyenne arithmétique sur un tel ensemble de valeurs. Les notes seront donc associées à une échelle ordinale permettant de calculer la valeur médiane d'un groupe.

Prédicats	Description
is_scale(s)	s est le nom d'une échelle
is_scale(u,s)	l'unité u est associée à l'échelle s
is_default(u,s)	s est l'échelle par défaut pour l'unité u
is_covering(s, t)	s autorise au moins toutes les opérations de t
is_legal(o,s)	l'opération o est autorisée par l'échelle s
is_operation(o,s,t,r)	associe l'op. o, les entrées s et t et le résultat r
is_lowest(s,t,r)	r est l'échelle la plus restrictive entre s et t

Tableau 4.6 – Prédicats essentiels à l'ajout les notions d'échelles de mesure au MKS

Les connaissances concernant les échelles de mesures sont ajoutées au MKS, en y ajoutant des prédicats et des règles pour l'analyse des échelles. Le tableau 4.6 présente les prédicats essentiels ajoutés.

is_default spécifie l'échelle de mesure par défaut associée à une unité. Par exemple, l'échelle par défaut pour toutes unités représentant une longueur est la ratio. L'échelle pour les températures en Kelvin est la ratio, celle pour les degrés Celsius et Fahrenheit est l'interval. Les échelles définies par défaut permettent de simplifier la mise en place d'une KR spécifique et permettent d'appliquer l'analyse des échelles en empêchant les erreurs les plus communes. Notamment, la définition d'échelles par défaut pour des grandeurs non physiques est nécessaire puisque ces grandeurs présentent le plus souvent des propriétés mathématiques limitées.

Le prédicat *is_covering* définit une relation d'héritage des propriétés d'une échelle à l'autre. Cette relation est définie dans le chapitre 2 et est représentée dans le tableau 2.1. Le prédicat *is_legal* spécifie les opérations autorisées par une échelle donnée. Ainsi, une opération peut être réalisée sur une grandeur, si et seulement si, l'échelle associée à la grandeur considère légale ou couvre une échelle qui considère légale l'opération.

Le prédicat *is_operation* s'évalue comme vrai, seulement si l'opération est légale pour les opérandes en entrées. L'échelle associée au résultat est définie comme étant l'échelle la plus restrictive des deux échelles d'entrées. La table 4.7 présente les règles associées aux prédicats présentés.

Transformation vers les échelles de mesure

Le prédicat *is_scale(u,s)* est la "tête" d'une règle qui consomme une expression d'unité et la transforme en l'échelle associée à l'unité. Tout comme la transformation vers la forme canonique (cf. 4.3.1), la règle se résout en visitant l'arbre syntaxique de l'expression. Lors de la visite de cet arbre, les

Tête	Corps
is_covering(s,s)	
is_covering(s,t)	is_covering(s,z) ∧ is_covering(z,t)
is_legal(o,s)	is_covering(s,t) ∧ is_legal(o,t)
is_lowest(s,t,s)	is_covering(t,s)
is_lowest(s,t,t)	is_covering(s,t)
is_operation(o,s,t,r)	is_legal(o,s) ∧ is_legal(o,t) ∧ is_lowest(s,t,r)

Tableau 4.7 – Règles associées aux échelles de mesure pour contraindre les opérations avec l'analyse des échelles.

sous-expressions sont analysées et leurs échelles évaluées en fonction des contraintes liées aux échelles sur les opérations.

Cependant, il n'est pas raisonnable de prédéfinir ou de dériver automatiquement les échelles associées à toutes les unités qui sont décrites dans le MKS. En effet, l'échelle de mesure associée à une grandeur est fortement dépendante du contexte, tout particulièrement de l'instrument de mesure employé pour relever la grandeur. Par exemple : relever des couleurs avec un colorimètre ou simplement en choisissant une couleur dans une liste. Dans le premier cas, la grandeur est une longueur d'onde avec un échelle ratio. Dans le second c'est une couleur avec une échelle généralement nominale (voir ordinale).

Pour pallier cela, le MKS permet à l'utilisateur d'entrer manuellement les échelles de mesure associées aux grandeurs. Ce point est expliqué plus en détail dans la section 4.4.

Exemple

Rappelons que P est une procédure qui implique de mesurer les longueurs et largeurs d'un toit, pour en calculer la surface, afin de définir le nombre de panneaux solaires qui peuvent y être installés (cf code 4.1). Le code 4.3 montre une requête d'analyse des échelles de mesure pour la procédure P. Dans cette analyse, l'échelle associée à la hauteur du toit et à la surface des panneaux est inférée à partir de leurs unités. L'échelle associée à la largeur du toit est le ratio, elle est imposée par la requête.

L'unification de cette requête propose de substituer S_out par l'échelle ratio, apportant à l'utilisateur l'information concernant les capacités algébriques de la grandeur résultante de la procédure de mesure.

Puisque la requête s'unifie pour l'analyse dimensionnelle et l'analyse des échelles de mesures, la procédure de mesure P est cohérente avec les

Code 4.3 Example of a scale analysis query

```
?- is_scale(meter, S_hauteur) ^
is_scale(are, S_panneau) ^
is_operation(mul, S_hauteur, ratio, S_toit) ^
is_operation(div, S_toit, S_panneau, S_nbr_panneaux)
```

règles de la métrologie. P a donc du sens dans le monde physique.

4.4 Vérification de procédures de mesures et langage de requêtes

La section 4.3 a montré comment spécifier et vérifier les dimensions et les échelles. Ces fonctionnalités sont étendues pour permettre de modéliser et d'analyser des procédures de mesures décrites par le MCM de la figure 4.1. Ces extensions réduisent l'écart sémantique entre l'architecte qui définit des procédures de mesures sous la forme d'assemblage d'agents et les concepts essentiels de la métrologie. Ces extensions définissent le langage de requêtes du MKS, le Measurement Knowledge Query Language (MKQL)

4.4.1 Abstraction des procédures de mesure

D'un point de vue architectural, les procédures de mesures sont décrites comme un ensemble d'agents interconnectés par des ports de communications (figure 4.3).

D'un point de vue métrologique, les agents sont des *instruments de mesure*, des *fonctions de mesure* et des *instruments indicateurs* (figure 4.1). Ceux-ci sont abstraits, par les unités des grandeurs d'entrées produites et de sorties affichées, ainsi que par les fonctions de transformations entre les ports. Chaque port de communication est abstrait par une paire $(dimension, échelle)$ pour permettre les analyses dimensionnelle et des échelles. En effet, les notions d'unités et de grandeurs n'ont d'intérêts pour les analyses que pour définir ces entités qui sont employées pour contraindre les fonctions constituant la procédure.

Le tableau 4.8 montre les prédicats essentiels utilisés pour l'abstraction des procédures de mesure. Le constructeur *port* assemble un d-vector et une échelle sous la forme d'une paire afin de modéliser le point de communication. Les prédicats *is_input* sont utilisés pour abstraire les instruments de mesure. Les prédicats *is_output* sont utilisés pour abstraire les instruments indicateurs. Les prédicats *is_function* sont utilisés pour abstraire les fonctions de mesure. Pour les instruments de mesure, l'échelle produite peut être dérivée automatiquement des unités utilisées par l'instrument ou alors elles peuvent être définies manuellement par l'architecte.

Le tableau 4.9 montre les règles utilisées pour appliquer les contraintes métrologiques aux modèles de procédures de mesures.

Ces règles permettent de vérifier que les unités aux entrées et sorties de la procédure de mesure sont correctement rédigées, quelles prennent en charge la transformation des unités en d-vectors et en échelles, quelles appliquent les règles des deux analyses au niveau de chaque fonction de

Fonctions	**Description**
port(d,s)	port appairant une dimension et une échelle
Prédicats	**Description**
is_input(u,p)	l'unité u est transmise au port p
is_input(u,s,p)	s est une échelle définie manuellement
is_function(o,p1,p2,p3)	op. o avec les ports d'entrées p1, p2 et le port de sortie p3
is_output(p,u)	le port p produit l'unité u

Tableau 4.8 – Prédicats utilisés pour abstraire les procédures de mesure sous la forme d'agents.

Tête	**Corps**
is_input(u,port(d,s))	is_dimension(u,d) ∧ is_scale(u,s).
is_input(u,s,port(d,s))	is_dimension(u,d) ∧ is_scale(s).
is_function(op,port(d1,s1), port(d2,s2),port(d3,s3))	is_valid(op,d1,d2,d3) ∧ is_operation(op,s1,s2,s3).
is_output(port(d,s),u)	is_dimension(u,d).

Tableau 4.9 – Règles pour l'application des contraintes métrologiques dans les procédures de mesure

Predicates	Description
is_trace(f,t)	définie une trace t vers un flux f
is_done(t)	la trace t est terminée
is_itrace(n,port(d,s),t)	instrument de mesure n et trace t
is_ftrace(n,op,port(d1,s1), port(d2,s2),port(d3,s3),t)	fonction n
is_otrace(n,port(d,s),t)	instrument indicateur n
is_checked_input(n, p, t)	entrée vérifiée n
is_checked_input(n, u, p, t)	
is_checked_function(n, op, p1, p2, p3, t)	fonction vérifiée n
is_checked_output(n, p, t)	sortie vérifiée n
is_checked_output(n, p, u, t)	

Tableau 4.10 – Prédicats pour la trace de la vérification

mesure.

4.4.2 Vérification des procédures de mesure

La cohérence avec la sémantique de la métrologie d'un modèle de procédure de mesure construit avec les prédicats et règles présentés peut être vérifiée. Ce modèle est donc appelé une *requête de vérification*. Si une substitution peut être trouvée par unification alors le modèle est correct. Sinon il est incorrect. Pour localiser précisément quel endroit du modèle est erroné, il faut un système de traces.

Le tableau 4.10 montre les prédicats utilisés pour générer la trace du processus de vérification du modèle. Les prédicats *is_trace* et *is_done* sont utilisés pour ouvrir et fermer un flux dans lequel sera écrie la trace. Les prédicats *is_itrace*, *is_ftrace* et *is_otrace* définissent les traces pour les entrées, fonctions et sorties. Les prédicats *is_checked_input*, *is_checked_function* et *is_checked_output* sont les contreparties capables de tracer des prédicats définis table 4.8. Ceux-ci proposent de tracer le processus d'unification et d'assigner des noms aux instruments et fonctions du modèle.

Le tableau 4.11 donne les règles utilisées pour vérifier et tracer les erreurs d'un modèle de procédure de mesure.

La combinaison des prédicats des tableaux 4.8 et 4.10 permet à l'architecte d'écrire une requête dont la vérification génère une trace pour les composants d'intérêt et est silencieuse pour les autres. Cela permet de fo-

Tête	Corps
is_checked_input(n,port(d,s),t)	is_itrace(n,port(d,s),t).
is_checked_input(n,u,port(d,s),t)	is_input(u,port(d,s)) ∧ is_itrace(n,port(d,s),t).
is_checked_input(n,u,s,port(d,s),t)	is_input(u,s,port(d,s)) ∧ is_itracet(n,port(d,s),t).
is_checked_function(n,op,port(v1,s1), port(v2,s2),port(v3,s3),t)	is_function(op,port(v1,s1),port(v2,s2),port(v3,s3)) ∧ is_ftrace(n,op,port(v1,s1),port(v2,s2),port(v3,s3),t).
is_checked_output(n,port(d,s),u,t)	is_output(port(d,s),u) ∧ is_otrace(n,port(d,s),t).
is_checked_output(n,port(d,s),st)	is_otrace(n,port(d,s),t).

Tableau 4.11 – Règles pour vérifier et tracer les modèles de procédures de mesure

caliser la vérification sur un ensemble d'agents ajoutés, ou modifiés, au sein d'une architecture construite avec des structures d'agents réutilisables ou déjà vérifiées.

Exemple

Le code 4.4 montre une requête qui permet d'appliquer l'analyse dimensionnelle et l'analyse des échelles de mesures à la procédure *P*. Rappelons que *P* sert à calculer le nombre de panneaux solaires à installer sur un toit en calculant sa surface (cf code 4.1). Ce code montre que le MKQL permet de simplifier la description de la requête qui était auparavent constituée de deux parties distinctes, une pour chaque analyse (cf codes 4.2 et 4.3), moins lisibles.

Code 4.4 Utilisation du MKQL pour simplifier la requête d'application des deux analyse proposée par le MKS à la procédure *P*.

```
?-
is_trace(fichier, t),
is_checked_input(instrument_largeur, mètre, P_largeur, t) ^
is_input(instrument_hauteur, mile, ratio, P_hauteur) ^
is_input(aire_panneau, are, P_panneau) ^
is_checked_function(aire_toit, mul, P_largeur, P_hauteur, P_toit, t) ^
is_checked_function(nbr_p, div, P_toit, P_panneau, P_nbr_panneaux, t)^
is_checked_output(resultat, P_nbr_panneaux, un, t) ^
is_done(t).
```

Notez que, en plus de proposer un seul code pour l'application des deux analyses, il est également plus compréhensible par un architecte logiciel non aguerri en métrologie. En effet, les mots-clefs employés sont proches du vocabulaire de la métrologie mais également de celui employé dans les approches composants ou agents. La requête modélise une procédure de mesure sous la forme d'agents. Chaque agent est une abstraction d'un instrument de mesure, d'une fonction de mesure ou d'un instrument indicateur. La communication entre les agents est réalisée à l'aide de ports qui abstraient les grandeurs pour chacune des analyses.

Ainsi, la modélisation d'une procédure de mesure se fait avec un vocabulaire proche de celui de l'architecte logiciel. La modélisation peut être compréhensible par un potentiel utilisateur final de l'application puisque les agents représentent soit des grandeurs à mesurer, soit des opérations sur les grandeurs, soit des grandeurs résultantes de la procédure de mesure. De plus, le modèle peut être utilisé pour vérifier sa cohérence avec la sémantique de la métrologie en le confrontant au MKS puisqu'il est également une requête d'unification pour la KR.

Avec un MKS contenant un système d'unités compatible avec le SI, il est possible de vérifier des procédures complexes. Le code 4.5 montre les équations d'une procédure de mesure *P2*. Le but de *P2* est de valider l'emploi du MKQL pour créer une requête d'analyse de procédure de mesure utilisant diverses unités, composées ou non, types de grandeurs et opérations. *P2* implique de mesurer deux longueurs et un temps. Les deux longueurs sont additionnées, le résultat est divisé par le temps pour trouver une vitesse. La vitesse est également divisée par le temps pour trouver une accélération. Les unités des longueurs sont le *mètre* et le *inch*. Le temps est en *seconde*. La vitesse est en *mile/mois* et l'accélération en *g* (g étant l'accélération de la pesanteur sur terre) et en *kip/kg*.

Code 4.5 Equations de la procédure de mesure complexe *P2* qui permet de tester l'expressivité du MKQL.

```
longueur1 + longueur2 = longueur
longueur / temps = vitesse
vitesse / temps = accel1
vitesse / temps = accel2

longueur 1 en mètre
longueur 2 en inch
temps en seconde
vitesse en mile/mois
accel1 en g
accel2 en kip/kg
```

Le code 4.6 montre la requête au MKS correspondant à *P2*, décrite à l'aide du MKQL. La figure 4.4 illustre ce code sous forme d'un modèle d'agents. Bien que cela soit possible, vérifier *P2* à la main n'est pas trivial. L'unification de cette procédure est réalisée par le MKS, assurant que la procédure est cohérente. Si l'on change une opération ou le type de grandeur d'une des unités de P2, la requête ne peut plus être unifiée, indiquant une erreur dans la procédure.

Code 4.6 Requête de vérification d'une procédure de mesure complexe.

```
?- is_checked_output(vitesse, quotient(mile,mois), P_v, t)
^ is_checked_output(accel1, g), P_a, t)
^ is_checked_output(accel2, quotient(kip,gram) P_a, t)
^ is_checked_function(vitesse_calc, div, P_l, P_t, P_v, t)
^ is_checked_function(accel_calc, div, P_v, P_t, P_a, t)
^ is_checked_function(add_calc, add, P_l1, P_l2, P_add, t)
^ is_checked_input(longueur1, mètre, P_l1, t)
^ is_checked_input(longueur2, inch, P_l2, t)
^ is_checked_input(temps, seconde, P_t, t)
```

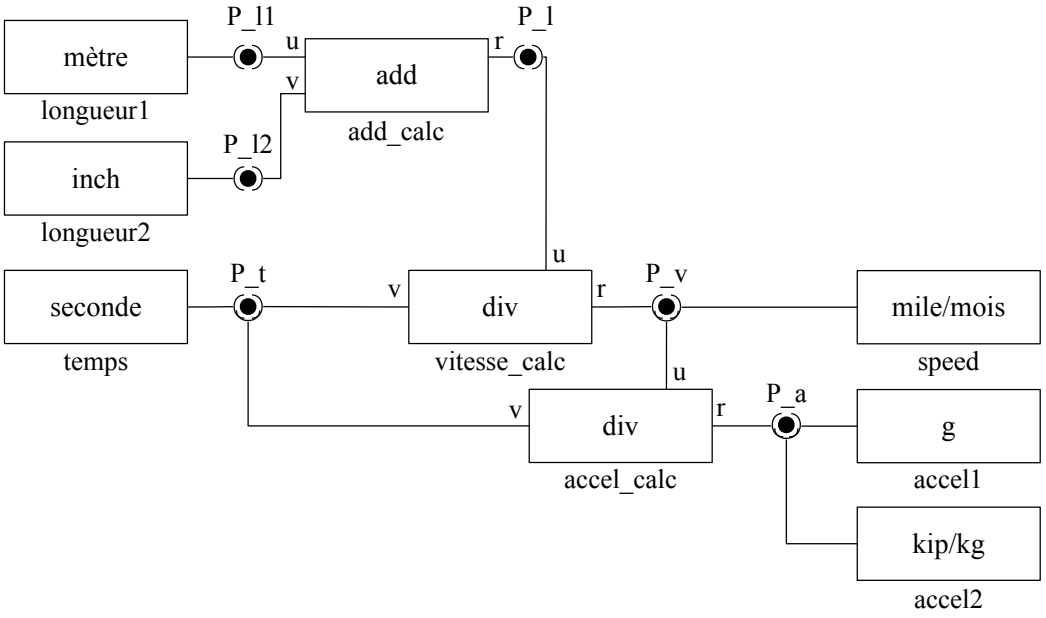

Figure 4.4 – Modèle d'agents décrivant la requête d'application de l'analyse pour la procédure *P2*.

4.5 Encodage des règles de conversion entre grandeurs

Les systèmes d'unités qui peuvent être inférés par le MKS sont décrits sous la forme d'un graphe d'unités interconnectées par des relations de conversions et d'alias. Ce graphe est décomposé en sous-graphes, chacun des sous-graphes regroupant l'ensemble des unités associées à une dimension du système.

Le chapitre 2 a montré que l'emploi d'unités hors du SI est très répandu et souvent source d'erreurs. Cette section présente des faits et règles qui une fois ajoutés au MKS permettent de prendre en charge les conversion entre les unités d'un système d'unités.

4.5.1 Prédicats de conversion

Chaque dimension est associée à une unité de référence. Cette unité de référence permet de définir le chemin entre les dimensions à l'aide d'opérations de compositions (mult, div). C'est l'unité de référence qui type la dimension. En effet, le prédicat *is_reference_unit(longueur, mètre)* déclare que la dimension *longueur* a pour référence l'unité *mètre*. Les autres unités du sous-graphe *longueur* ne sont pas liées directement à la dimension, mais

fonctions	**Description**
conv(ratio,r)	r est ratio de conversion
conv(interval,r,z)	r un ratio, z un décalage de zero
pair(v,w)	v et w des valeurs appairées
conv(nominal,lp)	lp une liste de paires <nom, nom>
range(v,w)	v et w des valeurs formant un intervalle
conv(to_ordinal,lp)	kp une liste de paires <range, nom>
conv(reciprocal,c)	c une fonction de conversion
conv(chain, cc)	cc une liste de conversions
Prédicats	**Description**
is_convert_to_base(u,c)	u est une unité et c une conversion
is_convert_from_base(u,c)	u est une unité et c une conversion
is_conv_applied_l(v1,c,v2)	v1 et v2 des valeurs, c une conversion
is_converted(v1,u1,v2,u2)	u1 et u2 des unités, v1 et v2 des valeurs
is_ok_converted(v1,u1,v2,u2)	identique à is_converted

Tableau 4.12 – Prédicats du MKS permettant l'inférence de conversion entre deux grandeurs.

à l'unité de référence directement ou par transitivité.

La figure 4.2 montre que des relations de conversion sont inférées entre toutes les unités d'un sous-graphe. Les relations de conversion sont, soit définies par des prédicats dans le MKS, soit déduites à partir de la transitivité des relations de conversion. Cette propriété de transitivité qui sera employée pour gérer les conversions.

Le tableau 4.12 présente les prédicats employés pour la description des conversions entre les unités d'un système. Rappelons que le MKS doit pouvoir prendre en charge les grandeurs non-physiques.

Les constructeurs *conv*, permettent de décrire la fonction de conversion qui prend place entre deux unités. Par exemple, la relation de conversion entre le degré Celsius et le degré Fahrenheit se décrit par le prédicat : *is_convertible(celsius, conv(interval, 1.8, 32), fahrenheit)*.

Ces fonctions de conversion sont identifiées avec l'échelle de mesure qui devrait être associée à la grandeur. *Ratio* définit le ratio de conversion entre deux unités. *Intervale* définit le ratio de conversion ainsi que le décalage entre les zéros de deux unités. *Nominal* définit les noms liés par la conversion. Par exemple, la conversion entre les stéréotypes de couleurs vues par deux individus peut comprendre les paires $pair(framboise, rose)$, $pair(rose, rose)$ et $pair(saumon, rose)$ rien que pour la couleur rose. Cette

conversion peut également être utilisée pour les grandeurs ordinales. *To_ordinal* définit une conversion d'une grandeur interval ou ratio vers une grandeur ordinal. Pour cela elle permet d'associer des intervalles de valeurs (*range*) à des noms. Ainsi une conversion entre mètre et une catégorisation des tailles des humains peut être définie avec les paires : $pair(range(0.1, 1.5), petit)$, $pair(range(1.51,\ 1.8),\ moyen)$ et $pair(range(1.81,\ 2.5),\ grand)$.

Reciprocal permet de représenter la réciproque d'une opération de conversion c. Ainsi, si c est la fonction *conv(ratio, 1.6)* qui permet de convertir des miles en kilomètres, alors *conv(reciprocal,c)* est équivalent à 1/1.6 et permet de convertir des kilomètres en miles. Normalement, la conversion *conv(reciprocal, conv(reciprocal, c))* devrait être identique à c. Cependant, les conversions *nominal* et *to_ordinal* peuvent induire une perte d'information qui empêche cette relation ; En se basant sur l'exemple des conversions de couleurs, la conversion de $\mathrm{C}_f(saumon)$ donnera $\mathrm{C}_h(rose)$. Cependant la conversion de $\mathrm{C}_h(rose)$ donnera $\mathrm{C}_f(framboise)$, $\mathrm{C}_f(rose)$ ou $\mathrm{C}_f(saumon)$ en fonction de l'algorithme d'unification.

Chain permet de regrouper une liste de conversions. La liste est interprétée sémantiquement comme une séquence de conversions à réaliser les unes après les autres. L'ordre dans la liste est primordial. En effet, une séquence de conversions doit être appliquée dans le bon ordre, sous peine de ne plus avoir de sens. Par exemple, pour convertir des Kelvin en degrés Fahrenheit, il pourrait y avoir la chaîne de conversion *conv(chain, [c1,c2])* avec c1 la conversion $\mathrm{K} \rightarrow^{\circ} \mathrm{C}$ et c2 $^{\circ}\mathrm{C} \rightarrow^{\circ} \mathrm{F}$. Prenons comme valeur à convertir 0K. En appliquant c1 puis c2 la valeur obtenue est -459.67F (la valeur correcte) alors qu'en appliquant c2 puis c1 la valeur obtenue est -241.15F.

Les prédicats de conversion permettent la création d'un graphe d'unités, connectant toutes les unités d'un système aux unités de référence de chaque dimension. Mais ces mêmes prédicats ne garantissent pas que des opérations de conversion ne soient pas, soit source de perte d'information, soit source d'imprécisions. En effet la première conversion perd des informations en passant un ensemble de valeurs vers une valeur unique. La seconde conversion est une source d'imprécisions car elle ne peut attribuer précisément une valeur unique dans l'ensemble de destination. C'est le même procédé que l'emploi d'un instrument de mesure et d'un instrument indicateur qui ne gèrent pas les mêmes degrés de précision sur les valeurs.

Les prédicats *is_convert_to_base* et *is_convert_from_base* permettent de définir la chaîne de conversion entre une unité et l'unité de base lui correspondant et inversement. Une unité de base étant l'unité de référence d'une dimension de base ou la décomposition d'une unité composée sous la forme d'une expression d'unité composée uniquement d'unités de base. Le mètre

est donc une unité de base pour la longueur. L'unité de base correspondant au newton pourra être par exemple *quotient(product(prefixed_unit(kilo, gramme) ,m), power(seconde, 2))*.

Le prédicat *is_conv_applied* est le prédicat qui permet de prendre en charge les conversions entre deux grandeurs. Il permet de définir que v1 est convertie en v2 par c.

Le prédicat *is_converted* permet de formuler une requête de conversion au MKS. En effet, ce prédicat prend deux grandeurs et infère qu'elles sont bien l'une et l'autre équivalentes après une conversion. Enfin, le prédicat *is_ok_converted* permet de formuler une requête de conversion en assurant sa faisabilité. Pour cela il vérifie que les deux unités sont bien de même dimension et qu'il existe donc une séquence de conversion entre les deux unités.

4.5.2 Règles de conversion

Le tableau 4.13 montre les règles associées aux prédicats *is_convert_from_base* et *is_converted* pour l'inférence de conversions entre deux grandeurs.

La notation *x is p* est employée ici pour simplifier la notation des clauses. Elle associe à la variable x le prédicat p. La notation *[h,t]* représente une liste d'éléments dont h est le premier et t la liste de tous les autres éléments.

Le tableau montre que les règles de *is_convert_from_base* et *is_converted* dépendent du prédicat *is_convert_to_base(u,c)*. Ce prédicat demande à trouver un chemin entre l'unité u et l'unité de base correspondante. Tout comme la canonisation (cf section 4.3) la conversion visite l'arbre syntaxique composant l'unité et génère la chaîne de conversions correspondante.

La conversion d'une unité vers sa base peut être vue comme une opération qui :

— pour chaque unité de base, cherche à remonter les arêtes vers la référence du sous-graphe correspondant,

— pour chaque unité composée, cherche les conversions vers les bases des unités qui la compose et les associe.

Les règles associées au prédicat *is_conv_applied* permettent l'application de toutes les conversions en fonction de leur type. Notez que les règles pour les conversions *reciprocal* ne sont pas détaillées pour des soucis de lisibilité du tableau. Celles-ci impliquent simplement d'appliquer la conversion réciproque à celle contenue.

Tête	**Corps**
is_convert_from_base(u,c)	is_convert_to_base(u,d) ∧ c is conv(reciprocal,d))
is_conv_applied(v,conv(ratio,r),w)	w is v*r
is_conv_applied(v,conv(intervale,r,z),w)	w is ((v*r)+z)
is_conv_applied(v,conv(nominal,pair(v,w)),w)	
is_conv_applied(v,conv(nominal,[pair(v,w),t], w)	
is_conv_applied(v,conv(nominal,[h,t]),w)	is_conv_applied(v,conv(nominal,t),w)
is_conv_applied(v,conv(ordinal,pair(range(x,y),n),n))	<(x,v) ∧ <(v,y)
is_conv_applied(v,conv(ordinal,[h,t]), w)	is_conv_applied(v,conv(ordinal,h),w)
is_conv_applied(v,conv(ordinal,[h,t]),w)	is_conv_applied(v,conv(ordinal,t),w))
is_conv_applied(v,conv(chain,[h,t]),w)	is_conv_applied(x,h,w) ∧ is_conv_applied(v,conv(chain,t),x)
is_converted(v1,u1,v2,u2)	is_convert_to_base(u1,c1) ∧ is_convert_from_base(u2,c2) ∧ is_conv_applied(v1, c, v2) ∧ c is conv(chain, [c1,c2])
is_ok_converted(v1,u1,v2,u2)	is_dimension(u1, D) ∧ is_dimension(u2,D) ∧is_converted(v1,u1,v2,u2)

Tableau 4.13 – Règles pour la conversion de grandeurs.

Le prédicat *is_converted* permet de convertir une grandeur g1 en une grandeur g2. Pour cela la chaîne de conversion c1 pour convertir g1 vers sa base est cherchée. La chaîne de conversion c2 pour convertir g2 vers sa base est également cherchée. La conversion de g1 en g2 implique de chaîner c1 et la réciproque de c2.

Cette approche n'est pas la plus rapide. En effet, elle oblige à déduire le chemin de conversion vers la base de chaque grandeur alors qu'il pourrait exister un chemin plus court. Cependant, il existe forcément une base pour chaque grandeur et cette base étant le point d'entrée de chaque sous-graphe, remonter vers elle est normalement réalisable à l'aide de la transitivité de is_convertible. Par contre, pour cibler directement une grandeur g2 à partir de g1 il faudra inférer un chemin de conversion qui, lui, risque d'être difficile, si ce n'est impossible à déduire dans un temps raisonnable, en fonction du nombre d'unités et de dimensions du système.

Enfin, le prédicat *is_ok_converted* vérifie que les d-vectors des deux unités de la conversion soient identiques. En effet, le prédicat *is_converted* trouve la chaîne de conversions pour chacune des unités vers son unité de base, mais ne vérifie pas que les bases soient identiques, ce qui peut permettre de convertir des grandeurs de nature différente.

Exemple

Le MKS permet de faire des conversions à l'aide du prédicat *is_converted*. Il est possible d'utiliser ce prédicat sur le système d'unités réduit présenté par le tableau 4.2 pour convertir des miles en kilomètres avec la requête :
is_converted(10, mile, X, km).
L'unification de cette requête substitue la valeur 16.09344 à la variable X. La chaîne de conversion se compose de : *conv(ratio,1.609344)*, *conv(ratio,1000)* et *conv(reciprocal,conv(ratio,1000)*, pour faire respectivement les conversions : $mile \rightarrow kilometre$, $kilomètre \rightarrow mètre$ et $mètre \rightarrow kilomètre$. Notez que la conversion contient deux opérations inutiles dans ce cas.

Ce système ne présente pas beaucoup d'unités et ses capacités de conversions sont donc assez faibles. Pour aller plus loin, nous proposons de convertir deux grandeurs de force, des kips en méganewtons. Pour cela, un MKS avec un système d'unités compatible avec le SI été développé. Il comprend les dimensions longueur, masse, temps, force, puissance, vitesse, aire et volume. À chacune de ces dimensions est associé un sous-graphe contenant les unités et les relations définies par les diagrammes d'unités disponibles sur le site web de l'entreprise Metric Methods^{SM}, une entreprise consultante spécialisée en métrologie [46]. Notez que la certaines des opérations

de conversions définies sur ces diagrammes ne sont pas exactes. En effet, les diagrammes employés proposent des unités qui sont définies hors du SI et dont les relations ne sont pas précisément définies. Ces inexactitudes de conversions peuvent être source d'imprécisions. Les calculs sur des nombres flottants réalisés par la machine ajoutent également des imprécisions.

L'unification de la requête :
is_converted(100, kip, X, prefixed_unit(mega, newton)
substitue la valeur 0.4448226 à la variable X. La chaîne de conversion se compose de : *conv(ratio, 1000)*, *conv(reciprocal, conv(ratio, 2.20462)* et *conv(ratio, 9.80665)* pour faire respectivement les conversions : *kip*→ *pound_force*, *pound_force*→ *kilogram_force* et *kilogram_force*→ *newton*. Le newton étant la référence pour cette dimension. Pour illustrer les imprécisions issues des conversions, nous pouvons convertir $1kip$ en *pound_force*, au lieu d'obtenir 1000, le MKS retourne 999.998.

Si maintenant on demande une conversion avec des expressions d'unités composées pour décrire une nouvelle force, non encodée dans le système la chaîne de conversions sera plus complexe. Par exemple : l'unification de la requête :
is_converted(1, power(mile,2), X, mult(km,km)
substitue la valeur 3097600 à la variable X. La chaîne de conversion se compose de : *conv(ratio,* 1.609344^2*)*, *conv(ratio,* 1000^2*)* et *conv(reciprocal, conv(ratio, 1000000)* pour faire respectivement les conversions : $mile^2 \rightarrow km^2$, $km^2 \rightarrow m^2$ et $km * km \rightarrow m * m$.

4.6 Utilisation de ProLog et conclusion

4.6.1 Utilisation de ProLog

Les différents prédicats et règles présentés dans les sections précédentes ont été implémentées avec la distribution SWI-ProLog [133] de ProLog.

ProLog permet d'encoder directement les prédicats et règles du MKS puisqu'il encode les connaissances d'un domaine en logique du premier ordre.

Trace de l'unification

ProLog propose un mécanisme d'écriture sur des flux (fichier, console). Ce mécanisme peut être employé pour tracer l'unification d'une requête.

SWI-ProLog propose des prédicats pour ouvrir et fermer un flux d'écriture ou de lecture d'un fichier. Il propose également un prédicat pour écrire sur un flux. À l'aide de ce flux, il est possible d'écrire pour chaque agent tracé, le résultat de son unification. Cela permet de remonter à l'utilisateur les dimensions impliquées à tous les niveaux de la procédure de mesure.

Nous rappelons qu'une procédure de mesure décrite par le Measurement Knowledge Query Language (MKQL) (cf sections 4.3 et 4.4) peut être vue comme un assemblage d'*agents* (cf figures 4.3 et 4.4. Chaque agent permettant de représenter une mesure de grandeur, une opération sur des grandeurs ou une indication d'une grandeur résultante de la procédure de mesure.

Notez que ProLog unifie les requêtes avec un algorithme d'unification à retour sur trace (backtracking). Un tel algorithme va parcourir tous les prédicats de la base, jusqu'à substituer toutes les variables de la requête. Si le prédicat est une règle, alors chaque prédicat de la règle sera soumis à cette unification. Lorsqu'une substitution s1 à une variable prouve un prédicat, l'unification continue à prouver les autres prédicats. Si s1 est incorrecte pour un prédicat dans la continuité de l'unification, alors l'algorithme d'unification fait un retour sur trace au moment où s1 aura été unifiée. Après ce retour, il cherche une substitution s2 et recommence la suite de l'unification. Ce cycle se répète jusqu'à trouver un ensemble de substitutions qui permette de substituer toutes les variables de la requête pour l'unifier, ou jusqu'à avoir testé toutes les substitutions possibles et déclarer que l'unification est impossible.

Cet algorithme implique que, lors de l'unification, les prédicats qui abstraient les entités de la procédure de mesure auront de multiples substitutions pour leurs variables avant de trouver celle qui est correcte ou non. Lors de l'analyse d'une procédure erronée, tous les chemins de canonisation possibles seront testés, toutes les substitutions de ports également. Ainsi, les informations tracées seront illisibles. Chaque retour sur trace implique des retours en arrière, qui réécriront des informations dans la trace à chaque nouvelle substitution testée.

Gestion des erreurs

SWI-ProLog propose un mécanisme d'exceptions qui permet de manipuler le processus d'unification. Ce mécanisme utilise un prédicat pour lancer une exception et un autre pour l'intercepter. Ainsi, lorsqu'une erreur est détectée dans l'unification d'une opération de la procédure de mesure, plutôt que de chercher à faire un retour sur trace et à ré-analyser des agents correctement unifiés, les agents tracés envoient une exception qui stoppe

	type d'agent	description de l'erreur reconnue
1	input	unité non conforme
2	input	échelle non conforme
3	input	dimensions de l'unité et du port différentes
4	input	échelles de l'unité et du port incompatibles
5	function	échec de l'analyse dimensionnelle
6	function	échec de l'analyse des échelles
7	function	erreur en provenance du port res
8	output	unité non conforme
9	output	dimensions de l'unité et du port différentes
10	input	erreur en provenance du port
11	function	erreur en provenance du port a
12	function	erreur en provenance du port b
13	output	erreur en provenance du port

Tableau 4.14 – Erreurs d'unification de requêtes remontées par le MKS.

l'unification, après avoir notifié le type d'erreur détectée dans la trace.

Néanmoins, il peut être intéressant d'autoriser le retour sur trace pour certaines analyses, comme pour le mécanisme de prescription présenté plus loin dans cette section (cf section 4.6.1). Rappelons que pour cela, lorsque le système n'est pas capable d'unifier un agent, l'agent substituera les dimensions et échelles de ses ports avec le message *error*. Les agents, qui tentent une unification avec un port qui contient le message d'erreur, l'indiquent dans la trace et indiquent qu'ils ont bien été unifiés. Ainsi, toutes les substitutions possibles seront testées sur les agents.

Le tableau 4.14 présente les différentes erreurs que nous avons identifiées. La partie supérieure du tableau présente les règles qui peuvent lancer des exceptions. La partie inférieure présente les règles qui sont employées pour gérer la prescription à l'aide de la diffusion de messages d'erreurs.

Prescription de types de grandeurs

Puisque ProLog est capable de chercher des substitutions pour toute variable du système, nous proposons également d'employer le MKS comme un outil de prescription capable de définir le type de grandeur qui pourrait être mesuré ou affiché par une procédure de mesure partiellement définie.

La multiplication et la division de grandeurs sont abstraites pour l'analyse dimensionnelle par le prédicat *is_valid(op,v,w,r)*. La logique des prédicats permet de remplacer théoriquement, n'importe laquelle ou toutes, les valeurs d'un prédicat par des variables.

Ainsi, il est théoriquement possible de proposer une requête : *is_valid(mul,* $<1,-2>$, W, $<1,0>$ dont l'unification substituerait $<0,2>$ à W. En prenant des d-vecteur $<monnaie, longueur>$ la requête permet de définir que : c'est avec la surface de panneaux solaires à installer qu'il faut multiplier le prix des panneaux pour une surface donnée, afin d'obtenir le prix de l'installation en résultat.

Cependant, l'analyse dimensionnelle manipule des d-vectors. Un d-vectors est un vecteur de nombres entiers. ProLog permet de substituer à une variable le résultat d'une opération de calcul de base (addition, multiplication puissance) sur des valeurs. Cependant, il n'est pas possible d'unifier une telle opération si l'un des opérandes est une variable.

SWI-ProLog propose une librairie qui utilise les concepts de la programmation logique par contraintes sur les domaines finis [34]. Cette librairie permet une unification tardive des variables impliquées dans des opérations algébriques. Elle est donc capable de définir l'ensemble des solutions à un problème algébrique partiellement défini.

Par exemple, on pourra trouver tous les résultats positifs inférieur à 10 de la somme 2B+B. Cela peut être réalisé avec par exemple la requête :
$+(A,B,R) \wedge -1 < R \wedge R < 10 \wedge A = 2B$.
L'unification de cette requête substitue à A la valeur $B*2$, à R la valeur $A+B$ et implique que la valeur de R est sur l'intervalle $[0,9]$. En remplaçant B par 2, 4 substitue A et 6 substitue R. Ainsi le seul résultat de cette opération est 6.

Il devient donc possible de résoudre des requêtes avec des unités inconnues. Cependant, il existe un nombre infini d'unités correspondant à une dimension. Pour simplifier l'unification et spécifier ouvertement qu'une unité n'est pas définie, nous proposons d'ajouter deux prédicats à la liste de ceux disponibles dans le MKQL :

— is_input_unknown(n,p,t), avec n un nom, p un port et t une trace,

— is_output_unknown(n,p,t), avec n un nom, p un port et t une trace.

Ces deux prédicats ajoutent à la trace le d-vector qui est contenu par le port.

Pour tester la prescription de grandeurs, nous proposons de reprendre la procédure *P2* de la section 4.4.2. Rappelons que *P2* est une procédure complexe qui a pour but de faire intervenir plusieurs opérations ainsi que plusieurs unités, complexes ou non (cf code 4.5).

Nous proposons de remplacer tous les agents d'entrées par des agents prescripteurs aux unités inconnues. Les prédicats *is_checked_input(n,u,p,t)* du code 4.6 sont donc remplacés par des prédicats *is_input_unkown(n,p,t)*.

L'unification de la requête produite est un succès. La trace résultante

affiche les d-vectors : $< 1,0,0 >$, $< 1,0,0 >$ et $< 0,1,0 >$ pour les agents longueur1, longueur2 et temps. Le système employé étant composé des dimensions $< longueur, temps, masse >$, les agents prescrivent d'employer respectivement des unités de longueur, longueur et temps. Ce qui est cohérent avec la procédure de mesure.

Il est possible d'obtenir des résultats moins attendus. Par exemple, avec l'opération $O\ i1 * i2 = mètre$, avec i1 et i2 des agents prescripteurs, l'unification substitue les d-vectors $< X, Y1, Z >$ et $< X, Y2, Y >$ à i1 et i2 et ajoute la contrainte Y1+1 = Y2. Cette unification montre que l'opération O est autorisée par l'analyse dimensionnelle si et seulement si, les dimensions de longueur et de masse sont identiques et la valeur de la dimension de temps de i1 est supérieure de 1 à celle de i2.

Réduction du champ des possibles

Le mécanisme de retour sur trace de ProLog permet de visiter l'ensemble du champ des possibles pour toutes les variables présentes dans une requête. Lors de l'analyse d'une procédure de mesure, plusieurs des étapes de l'analyse peuvent être achevées par des ensembles différents de prédicats. Le retour sur trace peut être inutile dans ces cas. Par exemple, la canonisation d'une unité demande de trouver l'unité de référence correspondant à chaque unité qui compose l'expression de l'unité. La transitivité et réciprocité des conversions des unités impliquent qu'il existe potentiellement un nombre infini de chemins pour trouver l'unité de référence. Par exemple, l'unité de référence du mile peut être trouvée par les chemins : $mile \rightarrow km \rightarrow m$, $mile \rightarrow yard \rightarrow m$, $mile \rightarrow yard \rightarrow mile \rightarrow km \rightarrow m$ ou encore $mile \rightarrow kilomile \rightarrow milikilomile \rightarrow kilomilikilomile \rightarrow etc...$

ProLog propose le mécanisme de contrôle de l'algorithme d'unification appelé *cut*. Placer un cut dans une règle, implique que tous les prédicats à gauche du cut ne doivent pas subir de retour sur trace, à moins que le retour n'implique une substitution définie avant l'évaluation de la règle.

Prenons une procédure de mesure *P3* qui implique de mesurer une longueur, une masse puis d'additionner ces deux grandeurs pour calculer un résultat. La longueur et la masse ne peuvent être associés qu'à une unique dimension et une unique échelle de mesure. Cependant, il existe plusieurs chemins de conversions permettant de trouver la dimension de la longueur et de la masse avec leurs unités. Ainsi, non seulement il n'est pas utile d'employer le mécanisme de retour sur trace pour modifier ces résultats, mais ce sera une opération consommatrice de beaucoup de ressources.

Afin d'empêcher le retour sur trace de chercher plusieurs solutions pour les entrées de *P3*, le code 4.7 utilise le prédicat *cut*. Lors de l'unification, les

deux premiers agents vont trouver les dimensions et échelles pour les grandeurs mesurées.trouver les échelles et les dimensions des unités mètre et gramme. Puis l'agent fonction tentera d'unifier l'opération d'addition sans succès à cause de l'analyse dimensionnelle. Et le retour sur trace qui s'initiera alors sera interrompu par le cut. Ce qui permet de prouver l'échec d'unification plus rapidement sans opérations inutiles.

Code 4.7 Exemple d'utilisation du cut de Prolog pour optimiser la vérification d'une procédure de mesure

```
is_input(mètre, P_longueur) ^
is_input(gramme, P_masse) ^
cut ^
is_function(add, P_longueur, P_masse, P_resultat)
```

L'ajout du cut permet de prévenir le retour sur trace de revenir sur les substitutions effectuées en amont. Ainsi, les unifications des deux premiers agents ne pourront pas être révisées par le retour. L'unification sera impossible avec ou sans le cut. Mais avec le cut, l'algorithme fera face à un ensemble de retours sur trace plus restreint et donc l'unification sera optimisée.

Il est cependant à noter que, la transitivité et la réciprocité des conversions pourraient potentiellement être utilisées pour décrire l'ensemble des unités qui correspondent à une dimension ; cependant, cet ensemble est infini. Il est possible de créer une infinité d'expressions d'unités pour un type de grandeur, simplement en ajoutant une composition puis son inverse. Ce point n'a pas été résolu par ces travaux pour le moment. Les cuts sont donc utilisés dans chaque agent pour éviter un retour sur le résultat proposé à moins que ce ne soit un résultat précédent qui soit revu.

4.6.2 Conclusion

Les différents prédicats et règles qui constituent le MKS viennent d'être présentés. L'intérêt principal de la structure proposée est la décomposition entre le système d'unités qui sert d'espace de raisonnement et l'inférence des analyses et du langage de requêtes.

En effet, une fois les analyses encodées, il est possible de vérifier une procédure de mesure, du moment que les unités qu'elle contient sont définies. Le MKQL permet de simplifier la définition des requêtes d'analyses, tout en proposant de remonter des informations sur les potentielles erreurs détectées lors d'une analyse.

Il est possible d'employer un MKS construit avec un système d'unités très restreint tel que celui présenté par le tableau 4.2. Mais il est également possible d'étendre ce système pour autoriser la modélisation et la vérification de procédures de mesure dans des domaines plus complexes.

L'intérêt étant donc de proposer un système modulaire. Un système d'unités restreint permet une analyse rapide, car le graphe des unités est petit et rapidement parcouru. Un système complexe permet d'exprimer plus de propriétés.

L'analyse des échelles permet d'inclure des grandeurs non physiques dans les procédures de mesures. Plutôt que de considérer ces grandeurs comme étant de type sans dimension, le MKS permet de leur attribuer une dimension (de base ou non). Cela induit que des grandeurs comme *nombre d'habitants par kilomètre carré* et *nombre de pots de peinture par décimètre carré* peuvent être différenciées et donc il ne sera pas possible de les additionner.

Il est à noter que les quatre opérations proposées peuvent également être spécifiées. En ajoutant une dimension *rotation*, il sera également possible de différencier le couple mécanique et une énergie ou encore une grandeur sans dimension et un angle. Cependant, un angle est défini comme étant issu de la division de deux longueurs ; il faudra donc spécifier les opérations de division et de multiplication pour permettre de prendre correctement en charge cette propriété. L'ajout de la règle suivante :
$is\_valid(div,l,l,a) \leftarrow is\_length(v) \wedge is\_angular(a)$
permet d'augmenter les capacités d'unification de l'analyse dimensionnelle pour considérer les rotations.

Nous n'avons pas exploré plus en avant cette capacité pour le moment : nos travaux s'étant focalisés sur la création d'un système dédié aux besoins industriels et non à l'exploration systématique de la métrologie. De la même façon, certains experts [78] déclarent que les longueurs devraient être exprimées en trois dimensions pour permettre des analyses correctes. En effet, nous assurons que la multiplication de deux longueurs implique une surface. En réalité, ce n'est pas le cas si les deux longueurs sont alignées.

Les analyses présentées dans ce document ne permettent que l'utilisation d'opérations de base. Mais la construction du MKS permet l'ajout d'autres opérations. Celles-ci permettent d'augmenter l'expressivité du MKQL et la précision des analyses.

L'approche présentée dans ce chapitre a été validée par la conception d'un système d'unités, compatible avec le SI et étendu avec des dimensions et unités hors du SI ainsi qu'avec des dimensions et unités pour des grandeurs non-physiques incluant : les couleurs, les booléens et les notes

d'examens. Les booléens sont particulièrement intéressants. Vus comme le type de grandeur capable de mesurer le résultat d'un choix ou d'une décision, ils permettent d'appliquer l'analyse dimensionnelle aux comparaisons (opérateurs relationnels) de grandeurs. Deux grandeurs sont comparables si elles sont du même type et le résultat de cette comparaison est de dimension booléenne. Les comparaisons sont des opérations très communes pour les procédures de mesure et sont indispensables pour des grandeurs non physiques associées à l'échelle nominale. Ensemble, ces différentes extensions produisent un MKS composé de plus de 200 unités, dimensions et alias qui impliquent des termes français et anglais.

Le MKS est donc hautement configurable. Il est possible de l'utiliser pour de petits systèmes d'unités avec un ensemble d'opérations dédiées à des domaines très spécifiques. Mais il peut également proposer un système d'unités et un ensemble d'opérations important, pour des utilisations plus génériques. Toutes les descriptions proposées pour le MKS sont déclaratives et résolues par unification.

Chapitre 5

Génie Logiciel pour le développement d'applications d'assistance à la mesure

« *Simplifier ce n'est pas faire simple.* »

Arthur Cravan

Sommaire

Le chapitre précédent a présenté le Metrology Knowledge Schema (MKS), une représentation des connaissances en logique du premier ordre dédiée à la vérification de la sémantique de la métrologie. Le MKS définit les différentes contraintes associées à l'utilisation de grandeurs pour assurer la possibilité de réaliser des transpositions correctes vers le monde empirique. De plus, il propose un langage pour la description de procédures de mesure. Toute procédure exprimée à l'aide de ce langage peut être interprétée comme une requête de vérification de la sémantique de mesure pour le MKS. Le MKS est aussi capable de réaliser des conversions entre toutes les unités d'une même dimension, peu importe leurs formes (base, alias, composée).

Cette approche permet donc d'assurer la conformité des procédures de mesure par rapport à la métrologie. Cependant, l'expressivité du langage est limitée à la description de procédures de mesure rudimentaires. Il ne considère pas les besoins liés à la réalisation de la procédure. En effet, un système logiciel permettant d'appliquer ces procédures de mesures nécessite non seulement la définition des éléments liés à la métrologie, mais également ceux liés aux interactions avec l'utilisateur et aux séquençages des activités.

Ce chapitre propose tout d'abord la description d'un langage de modélisation formel qui mélange les éléments de la métrologie, de l'ingénierie des processus et des interfaces utilisateur, le langage de modélisation d'application d'assistance à la mesure, Measure Application Modelling Language (MAML). Ce langage est dédié à la modélisation d'applications de mesures. Un éditeur capable de générer les modèles exprimés par ce langage est proposé. Ensuite, la conception d'une application configurable dédiée à la réalisation de procédures de mesure est présentée. Cette application est capable d'adapter son comportement en fonction de la procédure de mesure que l'utilisateur souhaite réaliser. Enfin, l'intégration de l'éditeur de modèles, du MKS et de l'application générique au sein d'une approche de développement logiciel semi-automatisée et outillée est présentée. Cette approche utilise les concepts de l'*Ingénierie Dirigée par les Modèles* (IDM), ou Model Driven Engineering pour utiliser le modèle de l'application comme source à vérifier et à implanter.

5.1 Langage de description d'applications de mesure

Les applications de mesure prennent en charge, habituellement automatiquement, les calculs impliquant des grandeurs et les déductions qui peuvent en être tirées (le prix estimé est, le meuble peut être placé à cet emplacement, allumer les radiateurs). Cependant, des interventions humaines sont souvent nécessaires en fonction du contexte de la mesure. C'est d'autant plus vrai dans un contexte de mobilité qui implique des prises de mesures non établies sur site. Cela implique des processus évolutifs car les méthodes changent en fonction de l'utilisateur et du contexte de mesure.

5.1.1 Spécifications du langage

Idéalement une application de mesure devrait assister l'utilisateur pendant ses activités de mesure. Ce qui implique de connaitre la séquence des mesures et comment sont réalisées les interactions avec l'utilisateur, ou les différents instruments.

Les procédures de mesure que peut exprimer le MKQL dans le chapitre 4 sont des spécifications des dépendances entre les grandeurs d'entrées, les manipulations des grandeurs et les grandeurs de sorties. Ces spécifications peuvent être utilisées pour décrire des applications de mesure basiques. Par contre, leurs comportements seraient certainement très éloignés des habitudes de mesure des utilisateurs. Le but du langage de modélisation MAML est de proposer une solution adaptée pour modéliser les applications de mesure.

Le MAML permet de modéliser des applications qui assistent les utilisateurs dans leurs tâches quotidiennes, incluant des activités de mesure. Ces utilisateurs possèdent des instruments dédiés à leurs méthodes de travail et sont habitués à les utiliser. Les applications doivent pouvoir apporter une plus-value à l'utilisateur, tout en évitant de lui demander d'apprendre de nouvelles méthodes ou d'acheter de nouveaux instruments. Pour être acceptées par de potentiels utilisateurs, les applications doivent, en plus d'être correctes, être configurées spécialement pour chaque utilisateur.

Cela implique des interfaces utilisateur intuitives qui permettent d'utiliser des valeurs issues d'instruments de mesure de l'utilisateur. Les instruments de mesure peuvent être de trois types :

— fourni par l'utilisateur, les valeurs sont entrées manuellement, souvent de manière textuelle, par l'utilisateur en utilisant des éléments d'interface,

— déclenché par l'utilisateur, les valeurs sont fournies par l'instrument, leur capture est activée par une interaction de l'utilisateur sur l'interface,

— continue, les valeurs sont fournies en continu par l'instrument.

Les applications doivent pouvoir s'adapter au rythme de l'utilisateur et ordonner les activités en fonction de l'utilisateur et des conditions environnantes. De plus, dans un contexte de prise de mesures sur site, des notes, des photos, des évènements de calendrier, des schémas peuvent être des informations importantes.

Ainsi, le MAML devrait permettre la description de modèles d'applications, composés d'activités de mesure ordonnées, d'interactions utilisateur et de capacités pour relever des informations autres que des grandeurs ; le tout adapté au contexte de la prise de mesures et à l'utilisateur.

5.1.2 Interactions et modèles de processus

Plusieurs formalismes existent pour modéliser, implanter ou exploiter des processus pour les systèmes d'information [20; 85]. Typiquement, les processus opérationnels, ou workflows sont des abstractions pour l'automatisation de processus. Ils font intervenir plusieurs activités qui traitent tout types de données qui sont ordonnées par des transitions [131]. Ce sont des abstractions dédiées à la modélisation de processus, qui incluent des interactions entre des activités réalisées par des humains et des machines[59]. Les concepts de base des workflows sont des *activités*, qui traitent des données à l'aide d'*actions* ; et des *transitions* qui autorisent le séquençage d'activités en fonction de *conditions*.

Les workflows sont capables de représenter tout type de processus et sont généralement à un haut niveau d'abstraction. Ils peuvent être spécialisés pour modéliser des procédures de mesure. En effet, les actions peuvent modéliser l'automatisation des fonctions de mesure ; les activités et les transitions peuvent quant à elles modéliser l'ordonnancement des mesures. Il est à noté que nous ne considèrerons pas les processus adhoc dans ce document. En effet, une procédure de mesure nécessite un ensemble de relevés de mesurages ; mais n'impose pas forcément une séquence de ces activités. L'approche proposée impose une séquence des activités.

Les Interfaces Humain-Machine (IHM) sont généralement construites en utilisant des composants graphiques simples et réutilisables appelés widgets [113]. Des widgets comme des champs de texte, des boutons, des affichages d'images, etc. peuvent être assemblés pour constituer des IHM complexes.

Ainsi, une IHM peut être une *vue* composée comme un arrangement de plusieurs widgets. Les éléments constituants la vue peuvent être soumis à une mise en page spécifique qui impose leurs positions. Les plateformes mobiles possèdent des écrans relativement petits et leurs IHM sont généralement épurées. Habituellement, les interfaces utilisateurs sont fortement dépendantes de la plateforme. Pour s'abstraire de ces dépendances dans le modèle, tout élément d'interface est un *interactor*. Les *interactors* sont composés pour construire des vues. Ils permettent des interactions en provenance de l'utilisateur et de capteurs. Ce sont les points d'entrée des interactions. Charge à l'application, dépendante de la plateforme, de définir les règles d'agencement et l'aspect des *interactors*.

5.1.3 Métamodélisation du language de spécification

L'IDM permet de représenter la structure et le comportement des systèmes avec des modèles (cf section 3.4). La création d'un DSL est une activité de métamodélisation nécéssitant l'usage d'un environement de métamodélisation. Le language MAML est formalisé avec le *Diagrammatic Predicate Framework* (DPF) workbench [72; 110].

Pour présenter les différents métamodèles en respectant les capacités de métamodélisation à plusieurs niveaux de DPF, la convention suivante est adoptée. Un *niveau Ni*, contient un *modèle Mi*. Mi peut être utilisé comme métamodèle pour la génération d'un *language de modélisation LMi*. La génération d'un LMi crée également un éditeur Ei. Un Ei permet de décrire des modèles Mi-1 ; conformes au Mi, qui appartiennent donc au niveau Ni-1.

L'architecture de métamodélisation du MAML est composée des modèles suivants :

— M4 est le métamodèle de DPF, proposé par défaut par le workbench, il est autodescriptif,

— M3 est le (méta)modèle utilisé pour décrire des processus interactifs,

— M2 instancie M3 pour ajouter les entités spécifiques aux procédures de mesure. M2 est le modèle qui définit le MAML,

— M1 est un modèle d'une application de mesure spécifique,

— M0 est la description de l'application qui assiste un utilisateur.

Métamodèle pour les processus intéractifs

Le métamodèle M3 combine des concepts issus des workflows [85] et ceux des interactions décrites dans la section 5.1.2. À ce niveau d'abstrac-

tion, les procédures de mesure ne sont pas représentées. Elles seront mises en place par une spécialisation de M3.

La figure 5.1 représente le M3 et ses relations avec le M4.

Conformément à DPF, le *DPF Metamodel* (M4) est utilisé comme métamodèle décrivant le LM4 qui permet de décrire M3. Ainsi il y a une relation entre chaque élément de M4 et l'un de ceux disponibles sur M3. Ces relations apparaissent sur le diagramme comme *:Node* ou *:arrow*. Elles sont omises quand elles sont évidentes. Le même principe est appliqué pour les (méta)modèles des niveaux inférieurs, qui sont décrits ultérieurement. Pour tous les liens qui ne présentent pas de multiplicité ; c'est la multiplicité "$*$" qui est sous entendue. Un lien avec plusieurs noms sous-entend l'existence de plusieurs liens identiques. Ils ne sont pas affichés par souci de clarté du diagramme.

Les transitions entre les activités peuvent être gardées par des conditions. Chaque activité est configurée comme une séquence d'actions. Les actions peuvent être déclenchées par des conditions. Le pont, entre les concepts de processus et ceux des interactions, est abstrait par les événements qui sont notifiés par les interactions.

Métamodèle du MAML

Le M2 instancie M3 pour apporter les différents types de données, les concepts de la métrologie, des workflows plus évolués, des éléments interactifs, etc.

Un extrait du M2 est présenté figure 5.2, pour illustrer comment cette instanciation est réalisée. Par exemple, le concept de données *Data*, introduit par M3 est employé pour obtenir les nouveaux concepts nombre (*Number*), grandeur *Quantity* et texte *Text*.

Les grandeurs possèdent deux membres ; l'unité et la valeur de la grandeur. L'entité *":Data"* présente sur le modèle implique que toutes les instances de données peuvent être utilisées par une opération de comparaison (*Comparison*).

Les *interactors* sont spécialisés pour obtenir des affichages de données (*Display*), des instruments de mesure de type déclenché par l'utilisateur (*user-trigger Sensors*), et des boutons pour déclencher des évènements : (*Button*).

D'autres *interactors*, plus évolués peuvent être proposés. Des *interactors* qui affichent des images, prennent des photos, permettent de dessiner, de sélectionner une valeur parmi plusieurs, etc. Ceux-ci permettent de proposer de meilleures interactions avec l'utilisateur ; ce qui améliorera son expérience.

Les actions sont utilisées pour définir les comportements autonomes de l'application. Ils manipulent, mettent à jour les données et automatisent les relevés d'instruments de mesure. La figure 5.2 montre comment les manipulations de données peuvent être associées, soit à des grandeurs, soit à des valeurs. Le lien en pointillés associé à un XOR est une contrainte DPF qui assure que a1, a2 et a3 sont tous de type *Number* ou *Quantity* pour une instance de *DataManipulation*.

Le lien *"then"* introduit par M3 permet de chainer les actions.

Pour définir le début d'une chaîne d'actions, une action *Start* est proposée. Ces actions détiennent une référence vers une condition *Control* et vers la première des actions à réaliser. Une fois la condition remplie, la chaîne d'action est évaluée. Des structures de contrôle plus complexes (boucle, réalisation conditionnelle, synchronisation) peuvent être ajoutées.

Notez que le M2 présenté propose des modèles pour des instruments de mesure (*UserTriggerSensor, GyroscopePolling*), des fonctions de mesure (*DataManipulation, Comparison*) et des instruments indicateurs (*Display*). L'ensemble permet à l'architecte de modéliser des procédures de mesure à l'aide d'entités proches du vocabulaire de l'utilisateur. En effet, l'extrait présenté ne montre qu'un instrument de mesure spécifique, le gyroscope. Mais en augmentant le nombre d'entités spécifiques, il sera possible de proposer des modélisations ad hoc pour chaque instrument proposé par l'utilisateur. Ce qui diminuera grandement la compréhension mutuelle entre l'architecte logiciel et l'utilisateur de l'application. Le *UserTriggerSensor* modélise un instrument de mesure générique. Le paramètre type permet optionnellement de spécifier l'instrument employé. Cette spécification permet d'augmenter la connaissance des instruments intervenant dans la procédure (cf section 5.3).

Le M2 est utilisé comme un métamodèle dans le DPF workbench pour créer le langage de modélisation LM2 et l'éditeur associé. ML2 est capable de modéliser toutes applications impliquant des procédures de mesures. Il est donc appelé le langage de modélisation d'application d'assistance à la mesure, Measure Application Modelling Language (MAML). La figure 5.3 montre un exemple de modèle décrit avec les concepts introduits par M2. Le modèle montre une activité qui active une action réalisant la mise à jour d'une grandeur angulaire. La valeur de cette grandeur est comparée à zéro pour définir si la garde de la transition est validée et permet d'enchainer sur une autre activité. Un instrument de mesure déclenché par l'utilisateur est également présent. Il permet de mesurer une longueur en utilisant un capteur de distance laser.

Le MAML présenté est généré à partir de la configuration essentielle de M2. Il est possible d'ajouter plus d'entités spécifiques au M2 pour augmen-

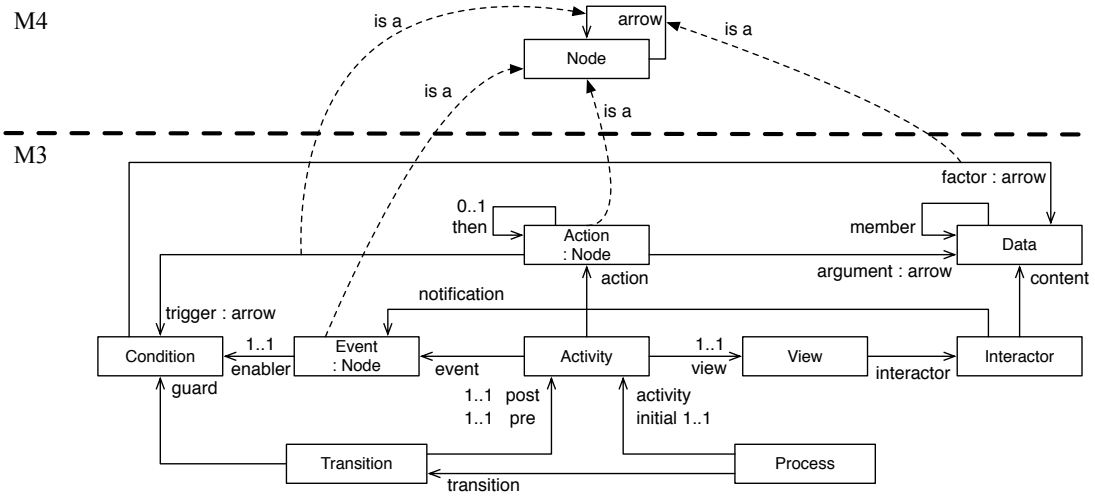

Figure 5.1 – Métamodèle pour les processus intéractifs (M3). La plupart des relations entre les entités du M3 et du M4 étant évidentes, elles sont omises.

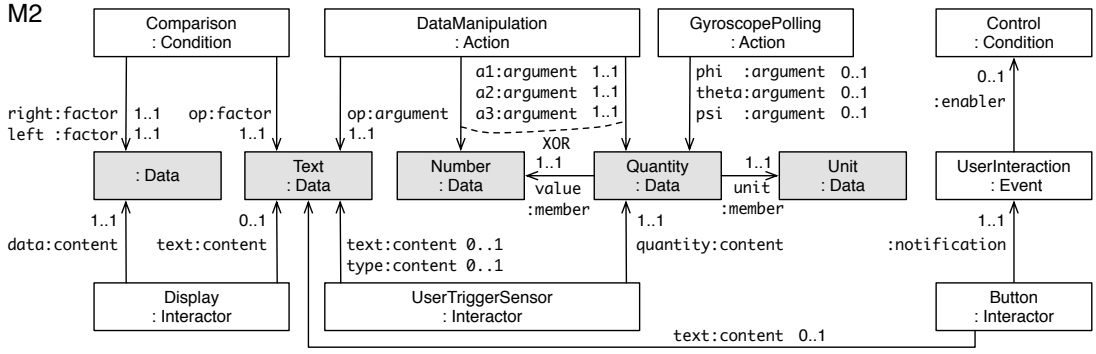

Figure 5.2 – Extrait du métamodèle du MAML décrit en utilisant le M3

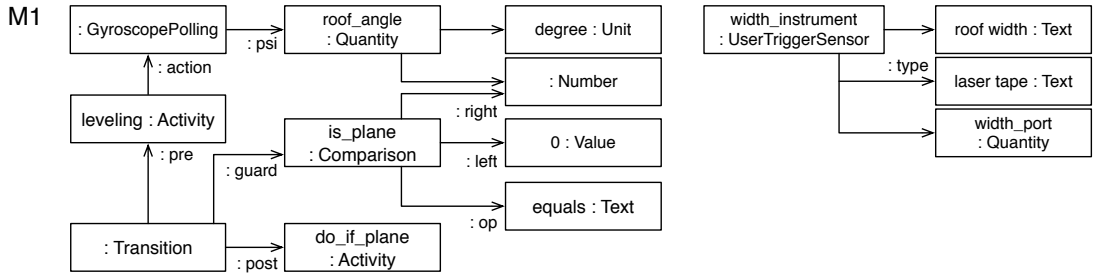

Figure 5.3 – Extrait d'un modèle décrit avec le MAML

ter l'expressivité du MAML. Cela augmenterait encore plus la compréhension entre l'utilisateur et l'architecte logiciel.

5.2 Application d'assistance à la mesure

Une application d'assistance à la mesure permet à un utilisateur de réaliser une procédure de mesure. Cette procédure dépend du domaine, des activités et des buts de l'utilisateur. Le but de cette section est de proposer une approche permettant l'implantation de ces applications dans une plateforme mobile.

5.2.1 Choix d'intégration

Pour proposer un système qui répond aux besoins spécifiques de chaque utilisateur il existe plusieurs alternatives :

1. proposer une application qui ne fournit que des interfaces pour utiliser des instruments de mesure ;
2. proposer une nouvelle application pour chaque procédure de mesure ;
3. proposer une application capable de configurer son comportement.

La solution (1) est la plus simple à réaliser. Mais c'est également la solution qui présente le moins de plus-value. Le système remplacerait au mieux un bloc note sur lequel les mesures sont notées avant de réaliser un traitement ultérieur.

Les solutions (2) et (3) présentent toutes deux des capacités équivalentes pour répondre aux besoins des utilisateurs.

(2) a l'avantage d'être une solution ad hoc pour un problème donné. Elle sera donc potentiellement optimisée (taille mémoire, vitesse de calculs) pour répondre à ce problème.

Par contre la génération et le déploiement d'une telle solution pour chaque procédure de mesure est plus difficile qu'avec la solution (3).

(3) est une application plus complexe et plus consommatrice de ressources. Elle doit contenir tous les éléments qui pourraient être utilisés dans toutes les procédures de mesure. Le fait de réaliser un comportement spécifique parmi un ensemble implique également des mécanismes de configuration et peut être d'animation de l'ensemble des éléments permettant la réalisation d'une procédure. Ces deux mécanismes ont également un coût mémoire et peut être en performances sur le système.

D'un autre côté, (3) est capable de proposer une bibliothèque de procédures de mesures. De plus, une nouvelle configuration est facile à ajouter au système. Il suffit de lui transmettre la description de celle-ci.

Pour les solutions (2) ou (3), il est possible de proposer une génération automatique à partir d'un modèle d'application (c.f. section 5.3).

Ces travaux présentent une implantation sous la forme de la solution (3). Cette solution est plus proche des canons actuels des applications pour plateformes mobiles. En effet, il sera possible de rendre disponible l'application et un certain nombre de procédures de mesure gracieusement tout en valorisant, selon la politique de l'entreprise, des activités de création de procédures faites sur mesure.

5.2.2 Métamodèle

Framework employé

Les agents permettent d'abstraire la complexité d'un système logiciel. Chaque agent réalise une tâche unitaire de manière autonome 3.2.

L'utilisation d'un ensemble d'agents logiciels permet une grande modularité et permet une cohérence dans le formalisme employé sur l'ensemble de ces travaux. Aux agents logiciels s'ajoute un ensemble d'objets organisationnels permettant de prendre en charge les séquences d'activités et les transitions.

Le système d'agents utilisé est dérivé de celui du framework PI [98]. Le framework est initialement conçu pour le pilotage en temps réel de systèmes automatisés. Ce pilotage est réalisé à l'aide d'agents, de leurs comportements et de leurs interactions.

PI propose un ensemble de concepts et une sémantique de réalisation qui permettent de prendre en charge les aspects structurels et comportementaux. Les aspect comportementaux sont décrit par des évènements d'une par asynchrone et d'autre part synchrone. Ces aspects sont nécessaires pour la définition de la structure de l'application, l'évolution des séquences d'activités de mesure et la gestion des instruments.

En utilisant PI nous assurons la réactivité du système et son évolution. L'ensemble des éléments de l'application de mesure sont des dérivés des agents proposés par PI.

Agents de mesures

La figure 5.4 montre la structure qui permet de modéliser les instruments de mesure, les fonctions de mesure et les instruments indicateurs.

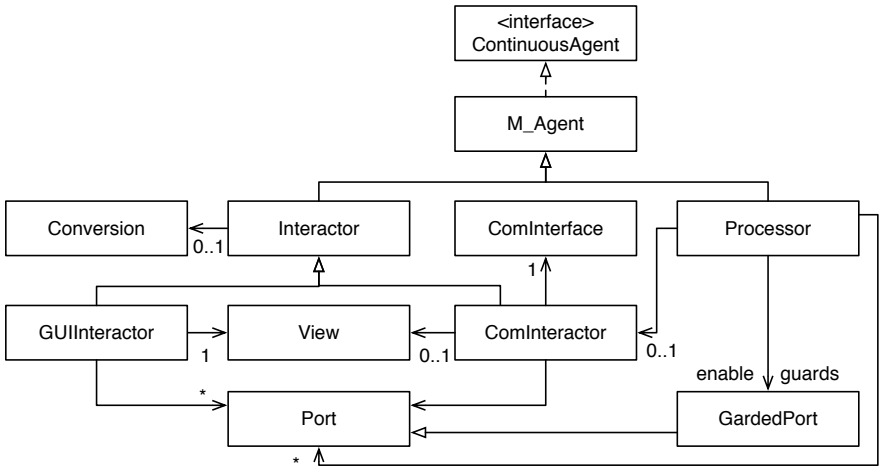

Figure 5.4 – Spécialisation des agents continus de PI pour l'application de mesure.

Les *M_agents* peuvent être vus comme un ensemble d'agents continus dont l'évolution est provoquée séquentiellement par le mécanisme de gestion d'évenements (société d'agents). Leurs comportement est réalisé en deux temps :

— analyse du contexte et mise à jour des sorties.

— analyse du contexte et mise à jour interne,

Les *Interactors* permettent de prendre en charge les interactions avec le monde extérieur. Ils sont utilisés pour recevoir ou afficher une grandeur sous la forme désirée par l'utilisateur. Les *GUIInteractor* gèrent des instruments *fournis par l'utilisateur* à l'aide d'une vue. Les *ComInteractors* prennent en charge des instruments *déclenchés par l'utilisateur* ou *continues* en employant des interfaces de communications. Ils peuvent également proposer des retours à l'utilisateur à l'aide d'interfaces.

Les *Processors* permettent de modéliser les différentes actions à réaliser lors des activités. Notez que, contrairement aux actions des modèles d'application, il n'y a pas de liens entre eux. Cette relation n'est pas nécessaire puisque les agents continus actifs du système évoluent tous sur ordre de la société.

L'évolution de chacun des processors est contrainte par une ou des gardes (*guards*). Lorsqu'il met à jour ses sorties, un processor le signale grâce au port *enable* qui servira de garde pour d'autres agents. Ce mécanisme permet de n'évaluer une fonction de mesure que lorsqu'elle le nécessite.

5.2.3 Agents d'activités

Les systèmes d'activités et de transition de la procédure de mesure permettent de guider l'utilisateur en limitant les informations à sa disposition et en contraignant son évolution dans la procédure [98]. La figure 5.5 présente les entités qui jouent ce rôle pour l'application mobile

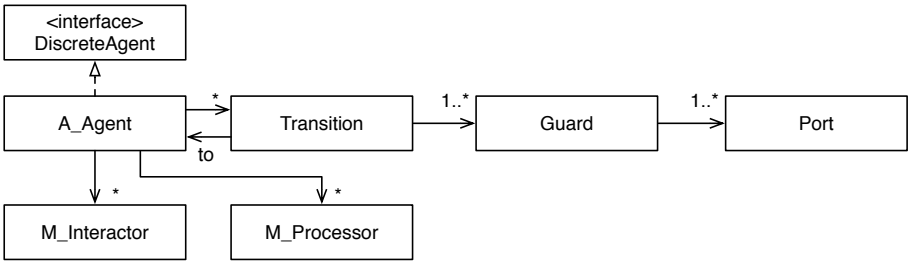

Figure 5.5 – Spécialisation des agents discrets de PI pour la réalisation d'un séquençage d'activités.

Chaque activité est représentée par un agent *A_Agent* qui est un agent discret. Les agents discrets sont évalués entre chaque évolution des agents continus. Lors de l'évaluation d'un *A_Agent*, il considère toutes les transitions qui peuvent mener à des activités suivantes. Si toutes les gardes d'une transition sont valides, alors la transition est passante et l'activité courante du système devient inactive et c'est l'activité suivante (*to*) qui devient active.

L'ensemble des *M_Agent* contenu par une activité est l'ensemble des agents qui sont actifs pendant cette activité. Seuls les agents actifs évoluent dans le temps.

Organisation des agents pour les vues

Chaque activité peut présenter une vue à l'utilisateur pour : le guider, entrer des mesures, activer des mesurages, passer à d'autres activités. La vue associée à une activité est issue de la combinaison des différentes vues que ses *Interactors* proposent.

La structure composée générée est proche de celle d'un modèle PAC (Présentation, Abstraction, Contrôle) [26]. La présentation de l'activité est représentée par la combinaison de toutes les présentations des sous-vues, auxquelles sont appliquées des contraintes de positions et de tailles. Il n'y a pas d'abstractions à ce niveau. Le contrôleur est la société d'agents qui aura pour but de faire évoluer les agents de l'activité.

Les *M_Agents* de l'activité sont des contrôleurs locaux. La présentation est une abstraction des vues ou interfaces de communication qu'ils pos-

sèdent. Ces présentations devront soit être mises à jour ou serviront de média d'informations en fonction de l'agent (instrument de mesure ou indicateur). L'abstraction avec laquelle le contrôleur interagit est l'association des ports de communications donnant sur l'extérieur de l'agent ainsi que des éléments internes à l'agent.

La figure 5.6 schématise cette architecture ainsi qu'une vue d'application résultante potentielle. L'activité présentée propose d'entrer manuellement une surface à peindre puis de récupérer le prix d'une peinture par web-service et affiche le prix pour recouvrir la surface définie. Un bouton permet de terminer l'activité.

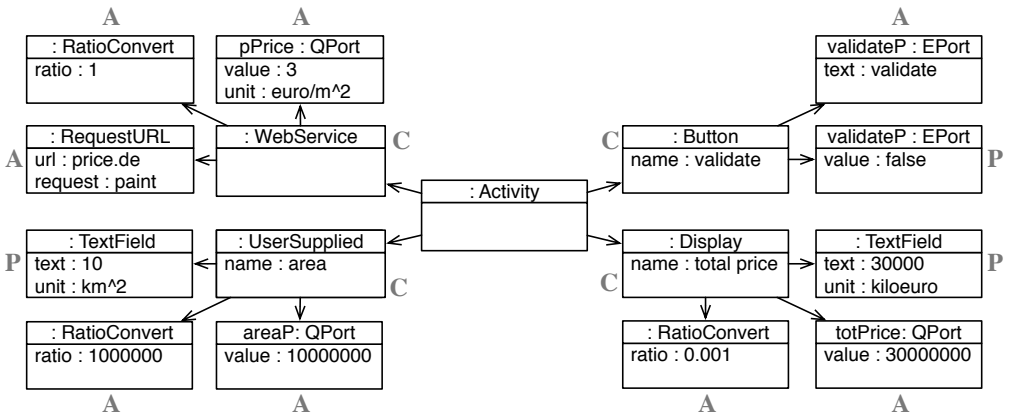

Figure 5.6 – Structure PAC pour la gestion des vues de chaque activités.

Notez que pour l'accès à des services web, capteurs communicants et autres utilisation des *ComInteractor*, nous considérons le modèle distant comme une présentation. Nous nous permettons ce raccourcis car c'est par le biais de cette communication que le contrôleur pourra accéder au monde extérieur à l'application. Cet accès permet de lire ou fournir des données à distance, tout comme une vue le fait localement sur l'écran de la plateforme.

Prise en charge des unités

L'utilisation d'unités dans les différents agents *Processors* demanderait :

— de transmettre des grandeurs dans toute l'application,

— d'adapter l'opération appliquée en fonction des unités d'entrées,

— de définir la ou les conversions à chaque opération.

L'avantage de ce procédé serait qu'avec un algorithme adapté, il serait possible de visiter la procédure de mesure dans sa globalité pour définir avec exactitude les opérations de conversion. Cela pourrait éviter certaines

conversions inutiles que notre approche implique. Mais en contrepartie, le système serait plus complexe à réaliser.

Plutôt que de complexifier la structure des différentes manipulations des grandeurs, nous proposons une approche d'homogénéisation des grandeurs dans le système. Pour cela, toutes les grandeurs, lorsqu'elles entrent dans l'application, sont converties automatiquement vers une unité de référence.

Ainsi, comme l'illustre la figure 5.7 l'application, d'un point de vue métrologique, peut être vue comme un ensemble de processors qui manipulent les données, qui sont mesurées ou indiquées par des interactors. Ces interactors sont en relation directe avec différents matériels qui utilisent différents formalismes de communication et encodages des grandeurs.

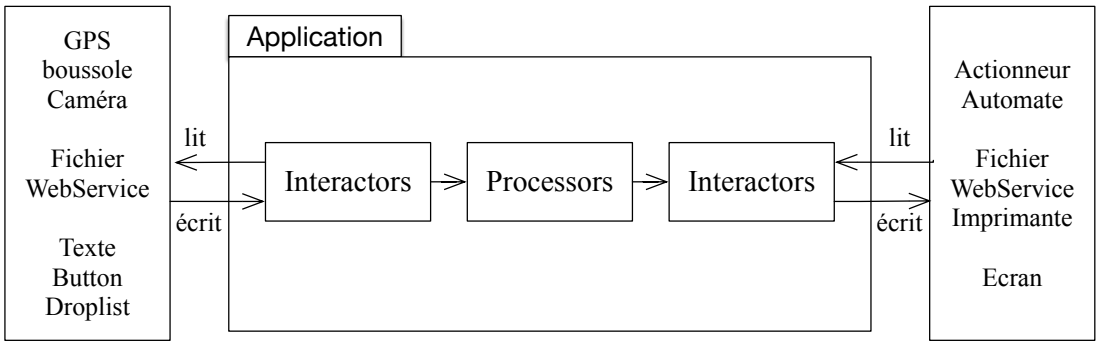

Figure 5.7 – Vue abstraite de l'extérieur de l'application. Les unités sont considérées comme homogénéisées dans l'application et doivent être traduites aux entrées et sorties du système.

Les Interactors jouent le rôle de traducteur. Ils capturent les données provenant du monde mesuré, les formalisent pour qu'elles correspondent au formalisme et à la synchronisation de l'application. Ils peuvent donc également prendre en charge la conversion des unités qui interviennent dans le système. Les opérations de conversion sont *simples* dans le sens ou elles doivent toutes avoir le même rôle. Pour les interactors en entrée, convertir une grandeur en une grandeur équivalente définie à l'aide d'une unité de référence. Pour les interactors en sortie, convertir une référence vers l'unité désirée pour la communication.

5.2.4 Configuration, construction et exécution

L'application est capable d'assembler un ensemble d'agents qui collaborent pour réaliser une procédure de mesure.

Il faut maintenant que l'application puisse réaliser la procédure de mesure d'un utilisateur. Pour cela, il faut être en mesure de décrire la confi-

guration des agents ; il faut pouvoir la construire et il faut faire évoluer les agents dans le temps.

La configuration d'une application peut être décrite par un modèle. Ce modèle sera nommé modèle d'implantation.

Le modèle d'implantation représente l'assemblage des éléments d'une configuration de l'application.

Le langage de modélisation correspondant est nommé le langage pour la description de configuration d'application de mesure, ou Executable Measurement Application Model Language (EMAML). Il décrit l'ensemble des configurations d'agents qui peuvent être instanciés par l'application. Sa syntaxe est donc fortement dépendante des capacités de l'application logicielle. La figure 5.8 présente le métamodèle du EMAML.

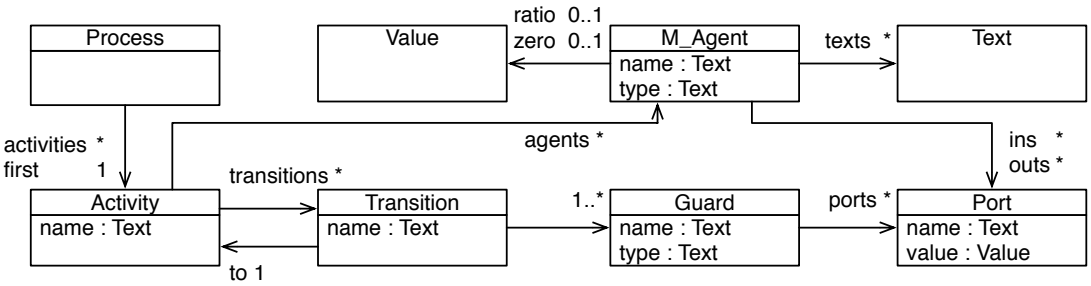

Figure 5.8 – Métamodèle représentant le langage EMAML

Rappelons que l'avantage des agents réside en leur capacité à fonctionner indépendamment les uns des autres. Un agent réalisant une addition ne se préoccupe pas de ce qui a produit les valeurs de ses opérandes, il a seulement besoin qu'elles existent.

La configuration de l'application peut donc être créée en décrivant les entités présentes dans le système puis les relations qui les lient. C'est la structure générée par ces relations qui va permettre de faire évoluer dans un ordre précis l'ensemble des agents.

Le code 5.1 présente sous forme d'expressions EBNF la syntaxe d'un langage EMAML basique.

Le but de cette syntaxe est d'être la plus générique possible afin d'autoriser simplement l'ajout de nouvelles phrases en fonction des éléments présents dans l'application. Avec la syntaxe présenté il est possible de générer un ensemble d'entités de bases, en décrivant leurs types et les relations de compositions.

Les éléments *superTyped* permettent de spécifier le type de l'entité à créer. Notez que les configurations d'applications seront générées automa-

Code 5.1 Description de la syntaxe d'un EMAML basique sous forme d'expressions EBNF.

```
<conf>   ::=  [<create>]+ ///// [<configure>]+ ///// <first>
<create> ::= [<super> | <superTyped>] <name>;

<configure> ::= <set> | <add>;
<set> ::= set(<name>, <name>, <param>)
<add> ::= add(<name>, <name>, <param>)

<first> ::= <name>;

<super> ::= Activity | Port | Transition
<superTyped> ::= [M_Agent Gyro] | [M_Agent LaserTape] |
                 [Guard Equal] | [Guard Smaller]

<param> ::= <val> | <text>
<name> ::= <text>
<text> ::= ["a"-"Z"|"_"]*
```

tiquement (cf section 5.3). Il n'est donc pas essentiel de contraindre les entités qui peuvent composées les *superTyped*.
<superTyped> : := <name> <name> pourrait, dans ce cas, remplacer les spécificités déclarées dans l'EMAML présenté.

La phrase *set* sert à définir la valeur d'un paramètre d'une entité.
set(gyro psi roofAngle), par exemple, permet de définir le port qui recevra les valeurs de l'angle psi d'un M_agent nommé gyro. La phrase *add* permet d'ajouter une entité à une liste de paramètres d'une entité. Comme par exemple une transition à une activité

first indique l'activité active au démarrage de la procédure de mesure. Il faudra donc l'indiquer à la société d'agents comme telle.

Attention, de par sa généricité, cette syntaxe autorise un grand nombre de phrases qui seront sémantiquement incorrectes pour l'application.
set(activity, psi, port) est une phrase correcte pour le EMAML. Cependant elle n'a aucun sens du point de vue de la configuration de l'application. Le but ici n'étant pas de proposer un langage exhaustif pour les configurations d'applications mais un moyen de communiquer des configurations qui seront générées à l'application.

Afin d'illustrer l'utilisation du EMAML, le code 5.2 présente la description des entités présentes dans la partie gauche de la figure 5.3. Rappelons que cet extrait de la figure représente deux activités de mesures *leveling* et

do_if_plane liées par une transition conditionnelle. L'activité *leveling* emploie un gyroscope embarqué pour mesurer l'angle de la surface sur laquelle est posé le mobile. Lorsque cet angle est nul, la transition est valide, ce qui déclenche l'activation de l'activité suivante.

Code 5.2 Description des entités de la figure 5.3 à l'aide du EMAML.

```
Activity leveling;
Activity do_if_plane;
M_Agent Gyro gyro;
QPort roofAngle;
QPort zero;
Transition t;
Guard Equal is_plane;
/////
set(gyro, dispName, 1);
set(gyro, ratio, 1);
set(gyro, zero, 275);
set(gyro, psi, roofAngle);
set(is_plane, right, roofAngle);
set(is_plane, left, zero);
set(zero, value, 0);
add(leveling, transitions, t);
set(t, next, do_if_plane;
add(t, guards, is_plane);
/////
t.add(do_if_plane);
t.add(is_plane);
leveling.add(t);
/////
leveling;
```

Notez la similarité entre les modèles décrits par MAML et EMAML. Bien que certaines relations ne soient plus présentes (unités)

La configuration représente les entités et les relations entre ces entités pour un système logiciel. L'application doit être capable de recevoir ces configurations, de les interpréter et de les exécuter.

Pour envoyer la configuration à l'application, différents systèmes existent. Elle peut être implantée directement dans l'application à la compilation, stockée dans la plateforme et cherchée à partir d'un explorateur de fichiers ou transmise par un webservice.

Une fois que l'application possède la configuration, elle doit l'interpréter. La sémantique associée aux différentes structures du EMAML est simple.

Toutes les entités *create* sont des appels pour créer une nouvelle entité. Cela implique de créer l'entité et à l'ajouter à une liste qui gardera en mémoire toutes les entités de la configuration.

Une fois que l'application a interprété la configuration, elle peut réaliser la procédure de mesure qui en découle.

Pour cela, le moteur d'exécution associé au framework PI est employé. Ce moteur permet de prendre un ensemble d'agents et de les faire évoluer dans le temps.

La différence avec les applications du framework PI d'origine [98] est que l'application est séparée en un ensemble d'agents actifs et un ensemble d'agents inactifs. En effet, tous les agents de la configuration sont paramètrés lors de son interprétation. Ils existent donc tous dans l'application à tout moment de son évolution.

Lorsqu'une activité est activée, tous les agents la constituant se voient ajoutés à l'ensemble des agents actifs. Lorsque l'activité est désactivée, tous ses agents sont inactifs. Seuls les agents actifs évoluent dans le temps à l'aide du moteur d'exécution.

L'application mobile étant présentée et ses capacités d'adaptation à une configuration précise présentées, tous les outils de la plateforme MEASURE (cf. figure 1.2) ont été présentés. Il faut maintenant permettre de les utiliser dans un même processus de développement.

5.3 Intégration des outils dans la plateforme MEASURE

Le MAML permet à un non expert en métrologie de modéliser des applications de mesure, correspondant à une description du comportement espéré par l'utilisateur. Les métamodèles, comme M2, décrivent souvent seulement la syntaxe abstraite d'un langage de modélisation. Ainsi, même si la définition de M2 assure la production de modèles corrects d'un point de vue syntaxique ou architectural, leur sémantique a besoin d'être définie et vérifiée. En effet, les modèles d'applications de mesure sont vraisemblablement sujets aux erreurs du point de vue de la sémantique de la métrologie.

Le chapitre 4 a présenté une approche pour vérifier les procédures de mesure à l'aide de requêtes de vérification. Les requêtes sont des abstractions qui peuvent être extraites des modèles d'applications. Plus précisément, les modèles d'applications sont des modèles opérationnels qui contiennent des activités, des chaînes d'actions, des conditions, des évènements, etc. Les requêtes de vérification sont des modèles déclaratifs qui

décrivent les relations entre les quantités. Transformer un modèle d'application en requête de vérification implique de supprimer les aspects opérationnels.

Il faut également apporter la sémantique applicative à ces modèles. Sous leur forme actuelle, les interactions avec le monde extérieur (affichage de valeurs, communication avec les instruments, etc) sont seulement énumérées (il y a des instruments de mesure, indicateurs). Les éléments liés à l'application manquent : comment une vue affiche les interactors ? qu'est-ce qui permet de faire évoluer le système ?

La section précédente a proposé un système logiciel capable de construire son comportement à partir de la description d'un processus à réaliser appelée modèle d'implantation. Ce système interprète le modèle, instancie la population d'entités logicielles correspondante et la fait évoluer dans le temps pour réaliser un comportement qui répond aux besoins de l'utilisateur. Le modèle d'implantation ne s'embarrasse pas des notions de grandeur. En effet, la procédure de mesure réalisée est considérée correcte, la gestion des unités est homogénéisée par des conversions décrites dans le modèle et les instruments ne sont que des médias qui fournissent (ou reçoivent) des informations de l'application. Transformer un modèle d'application en modèle d'implantation implique de supprimer les aspects liés à la métrologie.

La suite de cette section va présenter les deux transformations de modèle M2VQ et M2APP qui permettent de transformer un modèle d'application en respectivement, une requête de vérification et un modèle d'implantation.

5.3.1 La vérification d'applications

Transformation en requêtes de vérification

Le MAML implique une construction syntaxiquement correcte des modèles. Ils sont transformés en requête de vérification, pour définir et vérifier leurs sens [56] dans le domaine sémantique de la métrologie. Ces requêtes sont définie par le Measurement Knowledge Query Language (MKQL) qui a été décrit dans le chapitre 4.

Comme le présente la figure 1.2, un extracteur de requêtes, nommé le Model to Verification Query (M2VQ), génère des requêtes de vérification qui sont vérifiées par le vérificateur de conformité qu'est le MKS.

Bien qu'ils génèrent des modèles aux buts différents, les langages MAML et MKQL décrivent des procédures de mesure et les deux sont dérivés du modèle conceptuel (MCM) présenté. Ainsi, une définition de transformation utilisant les métamodèles de ces langages, respectivement M2 et le MKS,

peut être définie (figure 5.9).

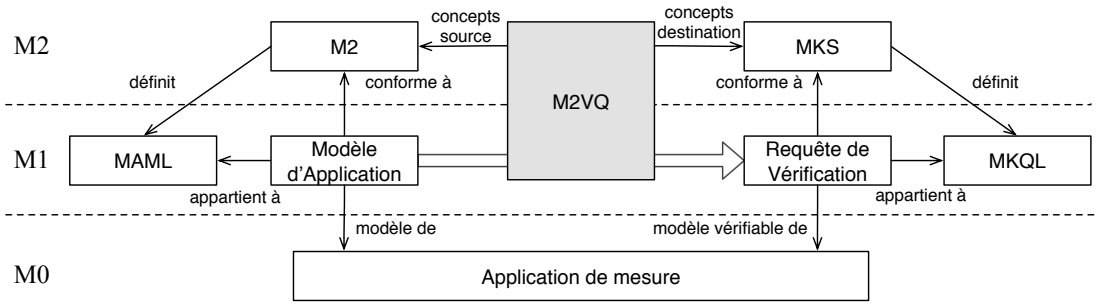

Figure 5.9 – Transformation du modèle d'application à la requête de vérification.

Le MKQL est un langage déclaratif dont la syntaxe est textuelle. Cela implique que la transformation vers une requête de vérification est un modèle vers texte, ou *Model to Text* (M2T). L'automatisation de cette transformation assure, en plus d'autres avantages, la qualité et la consistence du résultat produit [110].

Définition de la transformation

Les métamodèles source et destination partagent des concepts issus du MCM. Pour chaque languages, ces concepts sont spécialisés pour le rôle spécifique qui leur est attribué (modéliser, vérifier, appliquer). Quand les métamodèles de la source et de la destination partagent un concept du MCM, une relation doit être définie.

Les interactors représentant des instruments de mesure du M2, les actions représentant des instruments de mesure, les interactors qui définissent des grandeurs et les grandeurs constantes sont tous liés au concept d'instrument de mesure. Ce concept est abstrait par le prédicat *is_checked_input* du MKS. Les interactors d'affichage sont liées au concept d'instrument indicateur. Ce concept est abstrait par le prédicat *is_checked_output*. Les actions de manipulation de données qui traitent des grandeurs sont liés au concept de fonction de mesure. Ce concept est abstrait par le prédicat *is_checked_function*.

Certaines grandeurs du modèle M2 peuvent contenir des unités prescrites. Elles apportent des informations sur les grandeurs espérées pour cette grandeur précise. Elles devraient donc apparaitre dans la définition de la transformation. L'intérêt étant la création de requêtes les plus précises possible. Toutes ces grandeurs peuvent être abstraites par des prédicats *is_checked_input* ou *is_checked_output*. Rappelons que ces deux prédicats sont essentiels à l'application des analyses de la métrologie. Ils sont la tête

de règles qui considèrent une unité et la translate en un vecteur aux dimension et en échelle de mesure.

La transformation opère en visitant une première fois tous les noeuds grandeurs présents dans le modèle d'application. Si une grandeur a une unité prescrite, l'unité est considérée comme une constante et est abstraite par un instrument de mesure. Les autres grandeurs sont abstraites à des variables qui devront être substituées lors de l'unification de la requête. Dans un second temps, la transformation visite tous les noeuds qui sont des instruments de mesure, les manipulations de données et les affichages pour les transformer comme indiqué précédemment. Lorsqu'un tel noeud possède une caractéristique spécifique (type d'instrument) ou essentielle (unité de la grandeur), la transformation peut générer des connaissances supplémentaires. Ces connaissances ajoutent des redondances dans la vérification, ce qui consolide la requête.

DPF workbench propose un plugin qui étend le langage XPAND pour prendre en charge les transformation modèle vers texte, ou *Model to Text* (M2T) [110].

Le code 5.3 montre la définition de la transformation appliquée aux noeuds *UserTriggerSensors*. Rappelons qu'un *UserTriggerSensor* est un instrument de mesure générique dont seule la grandeur générée est un paramètre obligatoire. Le MAML permet également de spécifier le type de l'instrument employé. En connaissant le type de l'instrument, il est possible de définir le nombre et le type de grandeurs qu'il peut mesurer.

Le code 5.3 montre la définition de la transformation de tout *UserTriggerSensor*, ainsi que la transformation spécifiquement appliqué lorsque le type de l'instrument est un *laser tape* (ou capteur de distance laser).

Comme expliqué, cela permet d'injecter des connaissances de la métrologie complémentaires que le modèle n'exprimait pas ouvertement. Ici, la transformation injecte un prédicat *is_checked_input* qui va spécifiquement énoncer qu'un instrument de mesure de type *laser tape* produit une grandeur dont le type est *longueur*. Un gyroscope se verra proposer une définition qui associe à chacune des grandeurs produites une grandeur de type angulaire.

Cette capacité à injecter des connaissances additionnelles en produisant la requête simplifie la modélisation de l'application. En ajoutant progressivement de nouveaux éléments liés à la métrologie au MAML et à la définition de la transformation, la communication entre l'architecte et l'utilisateur s'améliorera et le processus de vérification se renforcera. C'est la responsabilité du créateur de l'outil de définir les connaissances additionnelles pour des interators spécialisés. Lorsque le MKS proposera la modélisation de capteurs, il sera possible de simplifier la transformation en retirant cette

Code 5.3 Définition de transformation d'un modèle d'instrument de mesure déclenché par l'utilisateur en une requête de vérification

```
FOREACH UserSuppliedSensor s
      "is_checked_input(" + name + ", " + unit + ", " + q_name + "
         _port, t) ∧"
      IF type == "laser tape"
              "is_checked_input(" + name + "_laser_tape, length, "  +
                  q_name + "_port, t)∧"
      ENDIF
ENDFOREACH

avec name = s.name()
       type = s.type().name()
       q_name = s.quantity().name()
       unit = s.quantity().unit().name()
```

injection de connaissances.

5.3.2 L'implémentation d'applications

La section précédente a déjà montré comment une configuration permettait de décrire le comportement d'une application. Elle a également brièvement montré comment un modèle d'application pouvait être converti vers un modèle de configuration.

Les entités présentes dans le modèle d'application et celles du modèle de configuration sont proches car les paradigmes employés pour les modéliser sont similaires.

Le modèle de configuration est un modèle épuré du modèle d'application dans le sens où il ne se préoccupe pas des notions de grandeur. La section précédente décrit comment sont prises en charge les unités des grandeurs pour simplifier leurs manipulations de manière homogène au sein d'une application.

La définition de transformation entre le modèle d'application et le modèle de configuration s'effectue entre le MAML et le EMAML. Son fonctionnement est similaire à celui entre le MAML et le MKS. La transformation visite les entités du MAML pour les transformer en texte représentant des créations, liaisons et compositions de la configuration.

De par l'organisation de la configuration, la transformation visite plusieurs fois les entités. Le code 5.4 montre la transformation d'un instrument de mesure du modèle d'application en interactor du modèle de configura-

tion.

Code 5.4 Définition de transformation d'un modèle d'instruments de mesure en une configuration pour l'application

```
//sensor creation
FOREACH UserSuppliedSensor s
      IF type == "laser tape"
              "LaserTape " + name + " " + name + " " + unit + " " +
                  ratio + " " + zero + ";"
      ENDIF
ENDFOREACH
//sensor links
FOREACH UserSuppliedSensor s
      IF type == "laser tape"
              name + ".out -> " + q_name + ";"
      ENDIF
ENDFOREACH

avec name = s.name()
       type = s.type().name()
       q_name = s.quantity().name()
       unit = s.quantity().unit().name()
       ratio et zero issus du MKS
```

Les mots-clefs *ratio* et *zero* représentent le résultat d'une requête de conversion faite au MKS. La requête est : *is_convert_to_base(unit,c)* unit étant le nom de l'unité associée à l'instrument de mesure du modèle. Le MKS prend donc ici une seconde utilité en décrivant les méthodes de conversion à appliquer aux grandeurs par les interactors

Cette transformation est appelée Model to Application (M2APP).

Troisième partie

Validation et Perspectives

Chapitre 6

Illustration de l'utilisation de la plateforme

« *Longue est la route par le précepte, courte et facile par l'exemple.* »

Sénèque

Sommaire

Cette section illustre le scénario du développement d'une application qu'un architecte rencontre en utilisant la plateforme MEASURE. L'application qui est développée est un assistant simple à l'estimation du coût d'installation de panneaux solaires.

Après cet exemple, l'utilisation de ces travaux dans le contexte industriel d'Actimage GmbH est présentée.

6.1 Spécification de l'application

L'application développée pour cette illustration a été imaginée avec Actimage GmbH. Son but était de faire avancer la réflexion autour de la thématique des travaux. L'application devait être simple, tout en comportant un certain nombre d'éléments qui ont été présentés :

1. les trois types d'instruments de mesure,
2. des grandeurs de différents types,
3. un système de grandeur réduit,
4. une grandeur hors SI,
5. des unités différentes pour un seul type de grandeur,
6. plusieurs opérations automatiques sur les grandeurs,
7. une opération permettant de valider la détection d'erreurs,
8. plusieurs activités,
9. une transition avec plusieurs gardes (automatique et manuelle),
10. un guide pour une mesure
11. une donnée non grandeur.

De plus, l'application devait servir de vitrine pour Actimage. Il fallait donc que la procédure de mesure soit une procédure à réaliser sur site et surtout, que le processus de développement de l'application à l'aide de la plateforme MEASURE soit compréhensible et utile.

Finalement nous avons décidé de réaliser une application qui servira à calculer le coût impliqué par une installation de panneaux solaires. Les besoins exprimés sont :

1. il faut pouvoir renseigner les informations concernant l'utilisateur,
2. la largeur du toit est mesurée en mètres manuellement,
3. la longueur du toit est mesurée en mètres avec un capteur laser communicant,
4. il faut pouvoir mesurer l'inclinaison du toit.

5. actuellement l'angle est estimé à l'aide d'un niveau à bulle. Une méthode plus précise n'est pas exclue si elle est simple et guidée.
6. il faut pouvoir estimer le prix de l'installation et l'afficher à l'utilisateur.
7. Le prix des panneaux solaires (tiling price) est exprimé en yards carrés installés par euro (ex. : $0.01 yard^2/€$)
8. L'inclinaison du toit implique un surcoût de 10 dollars par degré.

La solution proposée est une application qui présente deux vues, chacune correspondant à une activité spécifique. Sur la première vue, l'utilisateur peut renseigner des champs de texte pour entrer l'identifiant du client ainsi que la largeur mesurée du toit. Un troisième champ affiche que l'application est en attente d'une mesure pour la hauteur du toit. Mesure qui sera relevée par un capteur laser. Une fois que les deux longueurs ont été renseignées, un bouton permet de passer à la seconde vue. Le modèle de cette activité et de sa vue est présenté figure 6.1.

La seconde vue affiche le prix de l'installation qui est calculé à partir de l'aire du toit et du prix des panneaux. Ensuite, un tutoriel explique comment placer l'appareil mobile sur le toit pour mesurer son inclinaison. Cette mesure est déclenchée par l'appui sur un bouton de capture. La capture déclenche une chaîne d'actions qui lit l'inclinaison du toit à partir des données du gyroscope. Une fois l'inclinaison renseignée, la chaîne évalue le surcoût impliqué. Le modèle de cette activité et de sa vue est présenté figure 6.2.

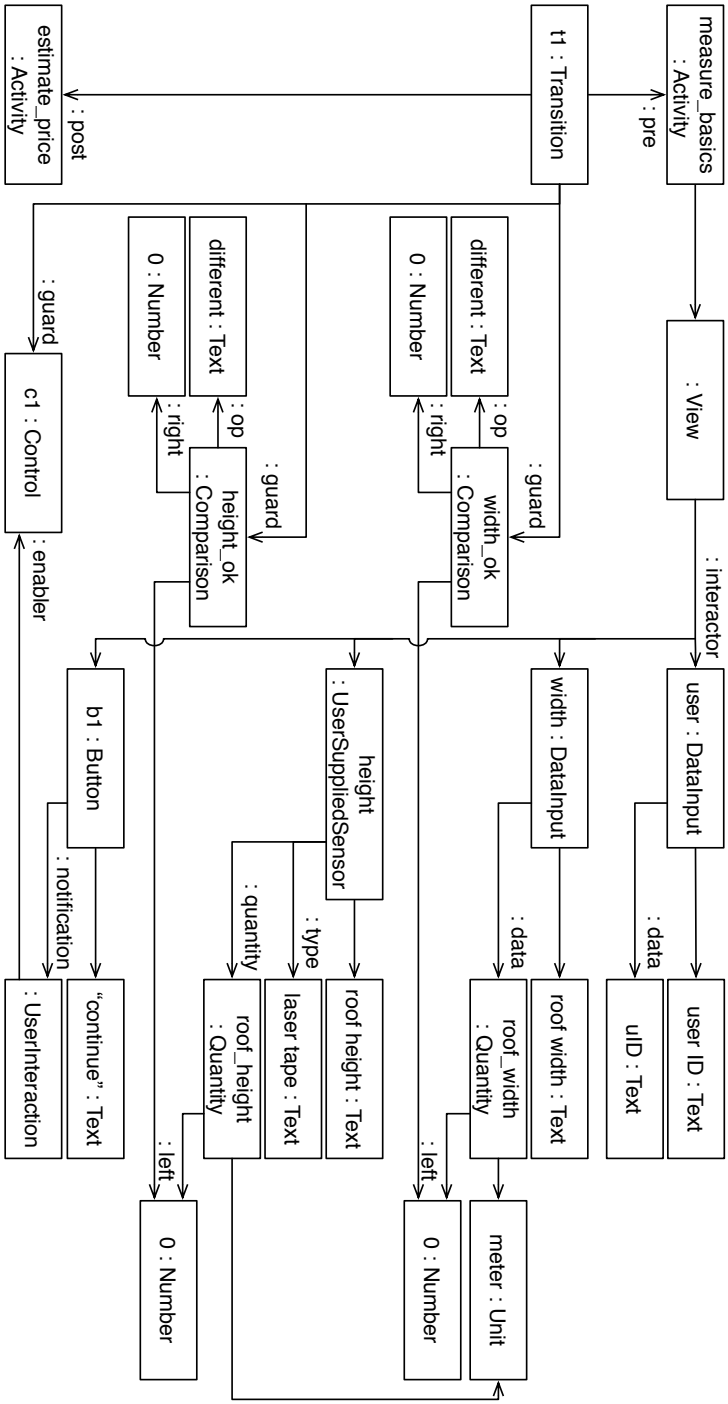

Figure 6.1 – Modèle de l'activité de mesure simple.

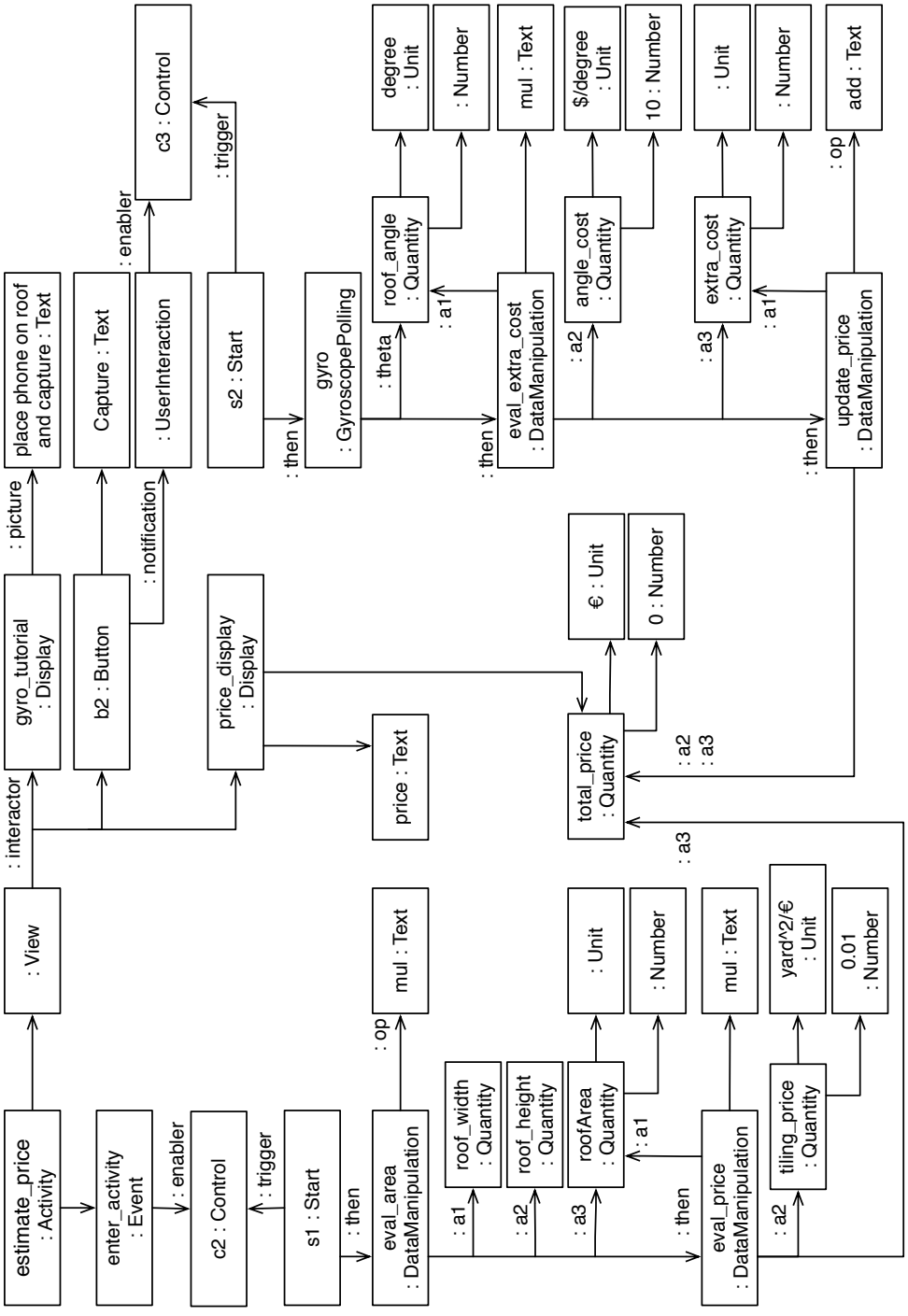

Figure 6.2 – Modèle de l'activité de mesure simple.

6.2 Vérification du modèle

Une fois l'application modélisée, il faut la vérifier. Notez que l'activité qui estime le coût (*estimate_price*) contient une erreur. En effet, *total_price* devrait être un prix, alors que l'action *eval_price* réalise l'opération suivante : $m^2 * yard^2/€$ ce qui produit des $longueurs^4/prix$. Cette erreur doit être détectée par la vérification.

Comme expliqué dans la section 3.4.2 ; une requête de vérification est extraite du modèle de l'application (code 6.1). Premièrement, les grandeurs avec des unités prescrites sont transformées en entrées. Ensuite, les instruments de mesure et indicateurs, ainsi que les fonctions de mesure, sont transformés en prédicats.

La transformation a généré des connaissances additionnelles pour les instruments de mesure laser et gyroscope. Celles-ci renseignent les types de grandeurs générées par l'instrument.

La requête est analysée pour identifier les erreurs sur les dimensions et les échelles. Comme il n'y a que trois dimensions d'intérêt dans l'application, les d-vector auront donc la forme <*longueur, monnaie, angle*>. L'action *eval_price* est de la forme :
$mul(<2,0,0>, <2,-1,0>) \rightarrow <0,1,0>$
ce qui est incohérent.

Pour pouvoir réaliser la vérification, les connaissances du domaine définies par le MKS contiennent les dimensions longueur, monnaie et angle ; les unités mètre, yard, dollar, euro et degré (angulaire) ; et les conversions mètre→yard, dollar→euro au minimum.

Lors de la résolution de la requête par unification, la trace rapporte une incohérence. L'instrument *price_display* requiert en sortie des dollars ($<0,1,0>$) et le port connecté contient le vecteur de dimension ($<4,-1,0>$. Donc, cette trace indique qu'il y a une incohérence au niveau de ce port. Pour corriger une incohérence sur un port, il faut se questionner sur la cohérence des entités directement connectées dessus. Dans ce cas, il y a plusieurs choix possibles :

1. modifier l'unité affichée par *price_display*,
2. modifier l'opération réalisée par *eval_price*,
3. modifier une unité ou opération plus loin dans la procédure.

C'est à l'architecte de faire le choix qui répondra le mieux à la situation. Pour cette application, le choix (1) n'est pas envisageable. Les besoins imposent d'afficher une valeur monétaire en sortie. Puisque le but est d'estimer un prix, il ne fait aucun doute que c'est ce qui est recherché. En

Code 6.1 Requête de vérification extraite du modèle d'application

```
is_trace(file, t)∧
// quantities with prescribed units
is_checked_input(roof_width, meter, roof_width_port, t)∧
is_checked_input(roof_height, meter, roof_height_port, t)∧
is_checked_input(tiling_price, quotient(power(meter,2),euro),
   tiling_price_port, t)∧
is_checked_input(roof_angle, degree, roof_angle_port, t)∧
is_checked_input(extra_cost, quotient(euro, degree), extra_cost_port,
    t)∧
is_checked_input(total_price, euro, total_price_port, t)∧
// measuring instruments
is_checked_input(width, meter, roof_width_port, t)∧
is_checked_input(height, meter, roof_height_port, t)∧
is_checked_input(height_laser_tape, length, roof_height_port, t)∧
is_checked_input(gyro_theta, degree, roof_angle_port, t)∧
is_checked_input(gyro_theta_GyroscopePolling, angular,
   roof_angle_port, t)∧
// measurement functions
is_checked_function(eval_area, mul, roof_width_port, roof_height_port
   , roof_area_port, t)∧
is_checked_function(eval_price, mul, roof_area_port,
   tiling_price_port, total_price_port, t)∧
is_checked_function(eval_extra_cost, mul, roof_angle_port,
   extra_cost_port, extra_cost_port, t)∧
is_checked_function(update_price, add, extra_cost_port,
   total_price_port, total_price_port, t)∧
// indicating instruments
is_checked_output(price_display, euro, total_price_port, t)∧
is_done(t).
```

revanche, les choix (2) et (3) sont cohérents. Avec (2) il est possible de changer l'opération en un quotient :
$quot(<2,0,0>, <2,-1,0>) \rightarrow <0,1,0>$
est une opération cohérente. Avec (3), modifier l'unité *tiling_price* est également un choix discutable. Ce n'est pas commun de rencontrer une unité comme celle-ci, habituellement ce sera plutôt le prix pour une aire donnée qui sera utilisé. Cependant, faisant partie des besoins exprimés par le client il convient de faire le choix (2).

Notez qu'il serait également possible de modifier la dimension présente sur le second port d'entrée de l'opération. Mais cela revient à réaliser des modifications qui sont de plus en plus éloignées de la source d'erreurs détectée. Dans une procédure de mesure, un port peut être connecté à plus de deux entités (instrument de mesure, indicateur ou fonction de mesure). Donc, plus la grandeur est éloignée du port dans lequel le problème est détecté, plus les risques de propager de nouvelles erreurs dans la procédure sont élevés. Il ne faut pas oublier que l'algorithme d'unification proposé ne détecte que la première erreur d'une procédure. Une fois celle-ci corrigée, de nouvelles erreurs plus en aval peuvent apparaître.

Une fois la correction établie, la requête issue de la nouvelle transformation s'unifie. Ce qui prouve que la procédure de mesure est correcte par rapport à la sémantique de la métrologie.

6.3 Implantation

Une fois l'application vérifiée, elle peut être transmise à l'application sous la forme d'une configuration.

Puisque toutes les opérations sur les grandeurs ont été validées par le MKS et que toutes les unités sont converties vers leurs références dans l'application, toutes les fonctions de mesures sont garanties d'être cohérentes.

L'application de la transformation M2APP correspondant au modèle de la figure 6.1 est décrite dans les codes 6.2.

Le résultat de la configuration est une application visuellement simple. La figure 6.3 montre les deux visuels obtenus.

L'application prend en charge un capteur de distance connecté. Ces capteurs permettent, une fois connectés à une plateforme mobile, de lui envoyer des informations sur une socket de communication. L'interactor prend en charge le protocole de communication. Une fois qu'il devient actif, il ouvre une socket de communication avec le capteur. Et quand une valeur a été communiquée, lors de sa prochaine activation l'agent lit la valeur, la convertit (ici la conversion n'a aucun effet) et la met à disposition

Code 6.2 Source de configuration issue de la transformation M2APP

```
Activity : measure_basics;
Activity : estimate_price;
DataInputText user userID;
DataInputQ width "roof width" meter 1 0;
LaserTape height "roof height" meter 1 0;
Button b1 continue;
transition t1;
Different width_ok ;
Different length_ok ;
Control c1;
TPort uID;
QPort roof_width;
QPort roof_height;
QPort zero;
/////
width.data -> roof_width;
height.data -> roof_height;
width_ok.left -> roof_width;
width_ok.right -> zero;
height_ok.left -> roof_height;
height_ok.right -> zero;
/////
measure_basics.add(t1);
measure_basics.add(user);
measure_basics.add(width);
measure_basics.add(height);
measure_basics.add(b1);
t1.add(estimate_price);
t1.add(width_ok);
t1.add(height_ok);
t1.add(c1);
/////
measure_basics;
```

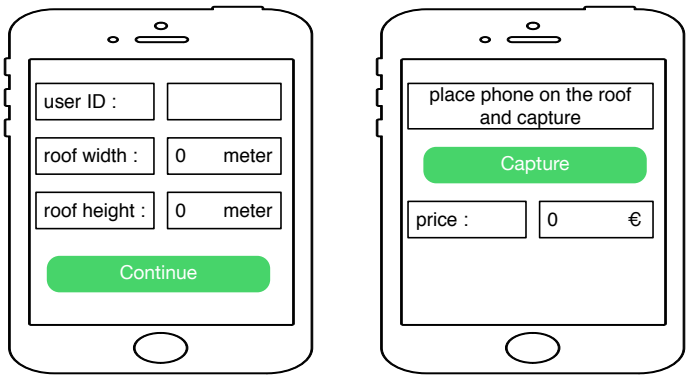

Figure 6.3 – Visuel de l'application auto-générée par la plateforme MEASURE.

dans le QPort connecté.

Un autre instrument de mesure est employé, le gyroscope embarqué dans l'appareil mobile. Une fois démarré il permet de mesurer l'orientation du mobile en flux continu. L'agent est un *ComInteractor* basé sur une structure PAC (cf. figure 5.6) dont le processeur requête un second PAC. Le second PAC est un agent qui gère en continu le capteur embarqué. Il prend en charge le démarrage de l'instrument et en capture les données en flux continu. Le processeur de l'agent *ComInteractor* quant à lui attend de recevoir le signal de l'utilisateur pour solliciter la valeur la plus récente du gyroscope au second processeur. Valeur qu'il convertira puis sauvegardera dans son abstraction. Cette composition est réalisable grâce à l'évolution maîtrisée des agents offerte par le moteur d'exécution du framework PI.

Notez que l'application peut être facilement améliorée en proposant un ensemble plus varié d'agents.

Un agent WebService pourrait contacter le site de vente des panneaux solaires pour obtenir une liste des panneaux et de leurs prix. Cela permettrait, à l'aide d'un affichage sous forme de liste, de choisir les panneaux en fonction de leurs prix. Cette communication pourrait être encore plus poussée en proposant de choisir les panneaux en fonction de leurs rendements et de leur coût d'installation. Le retour sur investissement de l'installation, qui est un très bon argument de vente, serait donc calculable. Ce mécanisme permet de mettre en place des fonctionnalités avancées.

La communication peut également être utilisée pour envoyer le devis de l'installation créé une fois qu'il a été signé par le client. Cela permet de lancer le processus d'archivage et de production plus rapidement ; sans risquer de perdre la commande.

En ajoutant une seconde base de données, il est possible de connaître l'ensoleillement moyen de la région. À l'aide du capteur GPS et de la bous-

sole embarqués, on peut récupérer, la position et l'orientation du toit. Ces informations permettent une estimation de la production électrique plus précise, en utilisant une base de données et deux capteurs embarqués dans tous les appareils mobiles actuels. La position permet également de simplifier le rapport concernant le client en associant à son adresse des coordonnées GPS.

Il est aussi envisageable de proposer une application qui permet de mesurer le niveau de l'utilisateur (débutant, habitué, expert) associé à la procédure de mesure. Cette mesure peut être prise en intégrant la vitesse d'exécution de l'utilisateur, en lui proposant d'entrer l'information manuellement, ou bien, la encore, celle-ci pourrait être associée aux informations de connexion de l'utilisateur. Prendre en compte le niveau de l'utilisateur, permet à l'application de proposer des activités adaptées pour être plus ou moins guidées.

6.4 Utilisation dans un contexte industriel

La section précédente a montré la puissance de développement offerte par la plateforme MEASURE. Cette section revient sur le contexte industriel du projet. Puis elle présente les développements qui en sont issus.

6.4.1 Contexte

Actimage GmbH a pour but le développement d'une flotte d'applications d'assistances aux mesures réalisées par des professionnels en situation de mobilité dans un premier temps et aux particuliers dans un second.

Ces applications ont pour but de devenir une boîte à outils personnalisée capable de profiter des capacités des systèmes mobiles pour :

1. partiellement remplacer la multitude de capteurs qui permettent de réaliser la procédure de mesure par des capteurs embarqués dans un système mobile,
2. guider l'utilisateur dans la réalisation de la prise de mesure,
3. réaliser des calculs fastidieux encore souvent effectués manuellement ultérieurement,
4. proposer des résultats in situ (estimations, contrat de vente, etc.),
5. intégrer plus aisément les résultats dans un système d'information d'entreprise.

La plateforme MEASURE est l'un des résultats de recherches réalisées autour de cette thématique. L'objectif original était l'analyse de la faisabilité

de l'utilisation de capteurs embarqués ou de capteurs communicants au sein d'une solution logicielle unique.

Depuis 2012, le monde du logiciel mobile a beaucoup évolué. Alors qu'à l'époque les premiers capteurs communicants grand public commençaient à faire leur apparition sur le marché, ils sont maintenant nombreux.

De plus, pour créer une application, chaque plateforme mobile nécessitait un développement spécifique. Aujourd'hui des technologies comme le moteur d'exécution Unity ou le langage Qt permettent de développer une application mobile qui pourra être utilisée sur toutes les plateformes mobiles. Dans ce contexte, il fallait pouvoir diffuser un logiciel capable de répondre à potentiellement l'ensemble des besoins de mesures de l'utilisateur, et ce quelle que soit la plateforme qu'il utilise.

Le projet qui servait de financement à ces travaux était un projet en partenariat avec la Hochschule de Offenburg. Ce projet visait à proposer un système logiciel capable de faire bon usage d'un capteur de distance développé par la Hochschule. Ce capteur est capable de capturer une vue 3D de l'espace environnant sous la forme d'un ensemble d'images contenant chacune 121 mesures de distances entre cinq et dix mètres. Chaque mesure étant précise à 2.5 millimètres près.

6.4.2 Choix et limitations de MEASURE

Le contexte décrit présente un champ des possibles extrêmement large. Plusieurs expertises sont nécessaires pour résoudre en entier les problématiques posées par le contexte. Que faire d'un nuage de points lors d'activités de mesures ? Quelles interfaces permettent d'en maximiser l'utilisation ? Quel est l'impact des incertitudes de mesures sur un ensemble composé de multiples captures (nuage de points 3D) ? Comment proposer une application adaptée aux besoins de tous les utilisateurs ? Qu'est-ce qu'une mesure ? Comment faire interagir les différentes grandeurs que peuvent relever les instruments de mesure ? Comment gérer les différents capteurs qui peuvent exister ? Comment favoriser le choix d'un capteur par rapport à un autre ?

La plateforme MEASURE a fait le choix de se limiter aux problématiques abordées dans ce travail de recherche afin d'y répondre, à notre sens, correctement. Les travaux qui sont présentés dans les chapitres précédents montrent la capacité de la plateforme à produire des modèles d'applications, dont le sens est garanti au niveau de la métrologie. Ces modèles sont transposables sous la forme de configurations d'un logiciel. Cette transposition est réalisée à l'aide d'un langage qui propose un ensemble de termes réduits et proches de ceux qu'emploie l'utilisateur de l'application.

6.4.3 Utilisation des travaux

Actimage GmbH [1] a utilisé ces travaux dans deux de ces projets récents : Datapipe et ActiNote 4.0.

Datapipe est un projet financé par le programme Eurostar de la commission européenne. Le but de ce projet est le développement d'un ensemble d'outils communicants qui permettront la semi-automatisation de processus d'analyse de données dans le domaine pétrolier (Oil & Gaz) [15].

Ce projet nécessite de pouvoir modéliser les processus sous la forme d'un ensemble de tâches potentiellement automatisées, qui devront être réalisées pour analyser les données. Les tâches peuvent être : l'accès à des données, leur décompression, l'application d'un algorithme, etc. Ces processus sont complexes et étaient entièrement gérés manuellement puisque leur modélisation requiert un ensemble de connaissances poussées dans la manipulation de données hétérogènes massives.

Datapipe propose un éditeur de modèle de processus qui permet de chaîner les différentes tâches à réaliser. De plus, une représentation des connaissances, ou Knowledge Representation (KR) encode les connaissances spécifiques au domaine de l'analyse de ces données. Elle permet de vérifier la cohérence d'un modèle de processus par rapport aux types de données qui y interagissent et par rapport à l'enchaînement de certaines tâches dont les dépendances sont clairement définies.

ActiNote 4.0 est la réalisation commercialisée de MEASURE. Il permet la modélisation, la vérification et l'implantation de processus d'assistance à des personnes dans des contextes de mobilité. Il est composé de trois outils : un éditeur, un compilateur et une application mobile.

L'éditeur permet la modélisation d'un processus à l'aide d'un langage graphique complexe. La syntaxe abstraite du langage est proche de celle de MEASURE. Par contre, l'éditeur proposé est plus avenant. Il permet la description des vues à l'aide d'une interface de composition de widgets qui représente l'écran du mobile tel qu'il sera présenté à l'utilisateur. De plus, les erreurs détectées dans le modèle sont affichées directement sur le modèle par un système de balises.

Le compilateur est un outil qui vérifie la syntaxe des modèles et leurs cohérences avec la sémantique propre à la plateforme. Il réalise également une transformation du langage vers une description d'une configuration de l'application mobile. Lors des développements préliminaires d'ActiNote, une version adaptée du MKS avait été intégrée au processus de compilation. Le compilateur transformait l'ensemble des actions du processus sous la

1. http ://www.actimage.de

forme d'une requête de validation de la procédure de mesure comprise dans le processus. Toutes les conversions d'unités impliquées dans un processus étaient prises en charge par le MKS.

Cependant, pour simplifier l'utilisation de l'éditeur actuel, le MKS n'est employé que pour ses capacités de prescriptions dans la version actuelle d'ActiNote 4.0.

L'application mobile est une application multi-plateforme qui est capable de recevoir et stocker des fichiers de configuration. Une configuration décrit l'ensemble des éléments qui sont présents pour générer un comportement conforme aux besoins d'un utilisateur spécifique.

Mais à part l'intégration de la sémantique de la mesure, l'ensemble d'Actinote 4.0 est une réalisation industrielle et commerciale dans laquelle sont utilisés les concepts de ces travaux.

Chapitre 7

Conclusion et perspectives

« *Car enfin, que sert-il d'écrire ? N'est-ce pas assez de penser ?* »

Cardinal de Bernis

Sommaire

7.1 Conclusion

Actimage GmbH[1] est un éditeur de logiciels proposant des solutions de prise d'informations en situation de mobilité afin d'en automatiser le traitement et le transfert vers les systèmes d'informations des clients. Dans ce contexte Actimage cherche à proposer des solutions logicielles qui assistent la prise de mesure, tout en étant de qualité et rapide à développer. Les travaux présentés ont été financés par Actimage pour proposer une solution à cette problématique.

Ces travaux ont présenté une approche de conception de logiciels d'assistance à la mesure en situation de mobilité. Les développements logiciels impliquant des procédures de mesure ont tendance à être développés sans forcément considérer les règles relatives au domaine de la métrologie. Malgré le sentiment de maîtrise du domaine que chacun possède de par son éducation, ce domaine est vaste et complexe. L'approche présentée propose une plateforme de développement logiciel qui augmente la qualité de systèmes impliquant des mesures. Pour cela, trois outils coopèrent grâce aux concepts de l'IDM.

Un système expert construit sur une représentation des connaissances de la métrologie ; le MKS. L'analyse dimensionnelle et l'analyse des échelles de mesure permettent d'assurer que des contraintes algébriques sur les grandeurs miment les relations du monde réel. En les appliquant sur une procédure de mesure, il est possible de garantir la cohérence des grandeurs mesurées et calculées avec les propriétés observées ou estimées. Le MKS encode ces connaissances en logique du premier ordre. Cela permet de raisonner sur l'ensemble des connaissances. Il encode la construction de systèmes de grandeurs, l'analyse dimensionnelle, l'analyse des échelles et la conversion d'unités. En plus de ces raisonnements, le MKS propose un langage de requêtes permettant la vérification de procédures de mesures. Ces requêtes sont décrites à l'aide d'un language de requête le MKQL. Il exprime des requêtes sous la forme de procédures de mesure constituées d'instruments de mesure, de fonctions de mesure et d'instruments indicateurs.

Un éditeur d'application de mesure spécialement conçu pour réduire l'écart sémantique entre un architecte logiciel et l'utilisateur de son travail. Cet éditeur propose à l'architecte d'employer un langage spécifique de domaine ; le MAML. Le MAML est décrit par des concepts de la métamodélisation et implanté avec l'outil DPF workbench. Il permet la création de langages de modélisation et d'éditeurs conformes aux langages. Leurs

1. http ://www.actimage.de

conformité étant prouvée par la théorie des catégories et des homomorphismes de graphes. Le MAML réunit des concepts de la métrologie, de l'ingénierie des processus et des interfaces utilisateur. Il permet ainsi la description de procédures de mesures outillées impliquant des interactions humaines. Le vocabulaire qu'il propose est constitué de termes proches de ceux qui permettent à l'utilisateur final de décrire ses procédures de mesures quotidiennes.

Une application mobile hautement configurable dédiée à l'assistance à la prise de mesure en situation de mobilité. L'application est construite à l'aide du framework d'agents PI. Ce framework permet la définition d'agents continus et réactifs dont l'évolution est garantie et maitrisée dans le temps. L'application mobile propose une librairie d'agents. Ces agents, une fois assemblés correctement, génèrent un comportement qui permet de répondre aux besoins de mesure de tout utilisateur. Un langage de description de configurations d'agents est proposé, le EMAML. Ces configurations peuvent être interprétées par l'application logicielle. Elle assemble les agents qui y correspondent pour générer un comportement spécifique. Un ensemble de configuration d'agents peut être stocké dans l'application, permettant de réaliser le comportement adéquat au contexte d'utilisation.

La plateforme MEASURE (cf figure 1.2) est dédiée au développement d'applications de mesures vérifiées. Elle propose un cycle de développement complet (spécification, vérification, implantation). Ce cycle est réalisé en utilisant les outils présentés (respectivement l'éditeur issu du MAML, le MKS et l'application logicielle. Pour cela, les concepts de l'IDM sont utilisés. Chaque outil fonctionne en produisant ou en utilisant des modèles de procédures de mesure spécialisés pour l'opération qu'ils réalisent. Ces modèles sont conformes à des métamodèles qui décrivent les langages de modélisation présentés (respectivement MAML, MKQL et EMAML). Afin d'automatiser le processus de développement, deux définitions de formations de modèles sont proposées. La première (Model to Verification Query (M2VQ)) permet de transformer les modèles d'application en requêtes de vérification ; la seconde (Model to Application (M2APP)) permet de transformer en configuration de l'application. Ces deux transformations permettent l'automatisation des processus de vérification et d'intégration à partir du modèle de description de l'application de mesure. La plateforme MEASURE permet donc de simplifier la spécification de ces applications ; de garantir la qualité des procédures de mesures impliquées ; et d'automatiser l'intégration de la procédure de mesure dans une application mobile.

L'approche est ensuite illustrée par l'utilisation de la plateforme pour développer une application, simple, permettant de mesurer d'évaluer le coût d'une installation de panneaux solaires sur une toiture. La vérification est

illustrée en incorporant volontairement une erreur dans le modèle de l'application. L'implantation du système est illustrée par des visuels de l'application générée. De plus, les capacités d'extensions de l'application sont expliquées. Enfin, une brève explication des réalisations industrielles dérivées de ces travaux est présentée. Les deux projets présentés sont Datapipe et Actinote 4.0 de l'entreprise Actimage GmbH.

7.2 Perspectives

L'ensemble des travaux présentés permet d'augmenter la qualité des applications de mesure et d'assurer que leur comportement soit conforme aux attentes des utilisateurs. Plusieurs axes d'évolutions sont envisagés pour approfondir les résultats obtenus.

Les résultats obtenus, avec les utilisations conjointes de l'éditeur de modèles et du vérificateur, sont prometteurs. L'axe qui sera donc prioritaire pour les travaux à venir est l'élaboration d'un langage spécifique de domaine embarqué (embedded DSL) [122]. Le but de ce langage sera le développement de logiciels impliquant des mesures vérifiées. L'avantage de cette approche sur l'approche actuelle est que les embedded DSL étendent les capacités expressives d'autres langages de programmation. Il serait donc possible de proposer une extension à l'un des langages de programmation très usités (Java, C++). Extensions capables de vérifier les procédures de mesure incluses dans les systèmes développés. Les contraintes liées à l'utilisation de termes spécifiques aux instruments de mesures et indicateurs seront limités par le langage. Avec une telle plateforme (embedded DSL + MKS), il serait possible de toucher une communauté bien plus large de développeurs. Et peut être même les sensibiliser aux problématiques de la métrologie.

Une autre perspective envisageable est l'ajout d'une nouvelle vérification dans le processus de développement. Pour le moment la sémantique de la métrologie proposée est celle liée aux analyses dimensionnelle et des échelles. Nous avons évoqué dans la section 3.4 l'importance des incertitudes de mesures dans un mesurage. Techniquement, un mesurage n'a pas de sens s'il n'est pas associé à une valeur d'incertitude qui permet de définir la fiabilité de la grandeur [88]. Vue l'étendue de ces travaux, il a été choisi de ne pas aborder cette thématique. Mais l'évaluation d'incertitudes de mesures, même sur un modèle, pourrait encore augmenter la fiabilité des résultats.

La transformation du modèle en requête de vérification implique déjà la prise en compte des différents instruments. Ce sont eux les principales

sources d'incertitudes dans une procédure de mesure. Une représentation des connaissances des instruments de mesure permettrait d'ajouter les notions de vérification des incertitudes. Elle préviendrait donc l'utilisation d'instruments incapables de réaliser des mesurages corrects par rapport à des tolérances d'erreurs.

Cette représentation des connaissances, ou Knowledge Representation (KR) des instruments de mesure pourrait également permettre de définir quels sont les instruments qui ne peuvent être utilisés pour obtenir un type de grandeur. L'intérêt serait double : premièrement, la transformation du modèle d'application vers la requête de vérification pourrait être allégée des spécificités liées aux instruments. Celles-ci étant apportées par la base de connaissances. Deuxièmement, il serait possible de proposer à l'utilisateur une liste des instruments de mesure qui répondent à ses besoins. En effet, le travail de métrologue implique également la définition des instruments de mesures intervenant dans une procédure de mesure. En connaissant les types de grandeurs fournies, l'incertitude de mesure des instruments et des notions de prix ; il devient possible de proposer un capteur qui sera optimisé pour une tâche donnée. Le MKS sera donc non seulement capable de prescrire le type des grandeurs qui sont attendues en entrée d'un système ; mais également les instruments qui peuvent être envisagés pour jouer ces rôles.

Bien que cela ne mène pas à de nouvelles recherches, l'intégration du système de vérification dans la plateforme Actinote 4.0 est également prévue. Cette intégration permettra de stresser le système pour en voir les limites ou les capacités. Nous n'avons en effet pas réalisé de tests à grande échelle sur le MKS. Cependant il faut toujours garder à l'esprit que les grandeurs produites par une procédure de mesure peuvent être considérées comme issues d'un instrument de mesure complexe. La vérification d'une procédure de mesure complexe revient donc à découper la procédure en plusieurs sous procédures. Chacune est vérifiée séparément avant d'être assemblée aux autres, jusqu'à arriver à la procédure globale. L'optimisation de cette approche de décomposition par rapport à sa confrontation avec le MKS peut s'avérer intéressante.

L'optimisation est un réel problème dans la vérification réalisée par le MKS. Actuellement seuls deux mécanismes permettant d'optimiser les unifications de requêtes de vérification sont utilisés. Le premier est l'utilisation des cuts qui permet de limiter la portée du retour sur trace. Le second est l'utilisation d'un système d'unités créé à partir d'un arbre dont les conversions ont, généralement, tendance à s'éloigner de la racine. Un chemin de conversion est donc assez simple à trouver en remontant ces conversions.

La précision des conversions souffre de cette structure. Toutes les conver-

sions entre deux unités $u1 \rightarrow u2$ impliquent de passer par l'unité de référence de la dimension $u1 \rightarrow reference \rightarrow u2$. Cela même s'il existe une conversion directe définie dans la base de connaissances. De plus, les conversions entre deux unités sont parfois approximatives, de par la définition de leurs unités. Un algorithme de conversion efficace devrait donc favoriser les conversions exactes. Cependant, les machines actuelles ne sont pas précises. Donc, si le chemin de conversion exacte implique plus d'opérations, il pourrait tout aussi bien être moins précis.

La RTM et l'analyse des échelles permet de prendre en compte des grandeurs non physiques. Notre approche considère ces grandeurs comme des dimensions supplémentaires pour le système. Nous n'avons que peu exploré les capacités apportées par ces dimensions supplémentaires. La métrologie "largement" définie (widely-defined measurement) [78] est une branche de la métrologie, hors des références, qui propose de considérer plus de dimensions pour chaque domaine. Par exemple l'utilisation de trois longueurs de types séparés pour chaque axe d'un plan à trois dimensions. Il semble intéressant d'approfondir nos travaux dans cette direction également.

Enfin, bien que nous employons la RTM, notre analyse des échelles est encore faible. Nous ne sommes pas capables de résoudre le problème du trajet en voiture cité figure 2.4. Pour rappel, il devrait être interdit de calculer une moyenne lorsque, pour une unité composée de la forme $numerateur/denominateur$, c'est l'unité du dénominateur qui est variable. Actuellement, l'analyse des échelles de mesure proposée est limitée à une simple répercussion des échelles proposées en entrée de la procédure vers ses sorties. Il faudra donc proposer des règles d'analyse plus poussées. Cela pourrait être réalisé avec des agents qui vérifient plusieurs opérations à la fois, pour avoir une plus grande connaissance de la procédure.

Bibliographie

Bibliographie

[1] 2010, «Modelica® - a unified object-oriented language for physical systems modeling - language specification - version 3.2», . 29

[2] Allemang, D. et J. Hendler. 2011, *Semantic web for the working ontologist : effective modeling in RDFS and OWL*, Elsevier. 43

[3] Almendros-JimÃ©nez, J. M. 2011, «A Prolog-based Query Language for OWL», *Electronic Notes in Theoretical Computer Science*, vol. 271, doi :http://dx.doi.org/10.1016/j.entcs.2011.02.008, p. 3 – 22, ISSN 1571-0661. URL http://www.sciencedirect.com/science/article/pii/S1571066111000454. 45

[4] Astarita, G. 1997, «Dimensional analysis, scaling, and orders of magnitude», *Chemical Engineering Science*, vol. 52, nº 24, doi : http://dx.doi.org/10.1016/S0009-2509(97)85420-6, p. 4681 – 4698, ISSN 0009-2509. URL http://www.sciencedirect.com/science/article/pii/S0009250997854206. 24

[5] Baader, F., éd.. 2003, *The description logic handbook : theory, implementation, and applications*, Cambridge University Press, Cambridge, UK ; New York, ISBN 978-0-521-78176-3. 42

[6] Barvik, P. 2014, *Model to Model Transformation Tool for the DPF Workbench*, thèse de doctorat, Department of Informatics, University of Bergen, Department of Computing, Mathematics and Physics Bergen University College, Norway. URL http://gs.hib.no/trac/eclipsep/export/1378/latex_sources/petter/Thesis/MasterVersion3.pdf. 61

[7] Bell, A. E. 2004, «Death by UML Fever», *Queue*, doi :10.1145/984458.984495, ISSN 15427730. 54, 55

[8] Berners-Lee, T., J. Hendler, O. Lassila et others. 2001, «The semantic web», *Scientific american*, vol. 284, nº 5, p. 28–37. 42

[9] Bindi, C. 2006, *Dictionnaire pratique de la métrologie*, AFNOR, La Plaine Saint-Denis, ISBN 2124607227 9782124607228. 26

[10] BIPM. 2006, «SI brochure, 8th edition - the international system of units (SI)», . 17, 20

[11] Bishop, J. 2007, *C# 3.0 Design Patterns : Use the Power of C# 3.0 to Solve Real-World Problems*, " O'Reilly Media, Inc.". 47

[12] Boehm, B. W. 1988, «A spiral model of software development and enhancement», *Computer*, vol. 21, nº 5, p. 61–72. 8

[13] Booch, G. 1986, «Object-oriented development», *IEEE transactions on Software Engineering*, , nº 2, p. 211–221. 46

[14] Botts, M. et A. Robin. 2007, «OpenGIS sensor model language (SensorML) implementation specification», *OpenGIS Implementation Specification OGC 07–000*. 27

[15] Bourgeois, F. et P. Arlaud. 2015, «Datapipe : A Configurable Oil & Gas Automated Data Processor :», SCITEPRESS - Science and and Technology Publications, ISBN 978-989-758-176-2, p. 75–96, doi :10.5220/0006163700750096. URL http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006163700750096. 145

[16] Boyd, T. D. et M. H. Jameson. 1981, «Urban and rural land division in ancient Greece», *Hesperia : The Journal of the American School of Classical Studies at Athens*, vol. 50, nº 4, p. 327–342. 16

[17] Brambilla, M., J. Cabot et M. Wimmer. 2012, *Model-driven software engineering in practice*, nº 1 dans Synthesis lectures on software engineering, Morgan & Claypool, San Rafael, Calif., ISBN 9781608458820 9781608458820 9781608458837. 8, 52, 57, 62, 64

[18] Bratko, I. 2001, *Prolog programming for artificial intelligence*, Pearson education. 35

[19] Bruneliere, H., J. Cabot, F. Jouault et F. Madiot. 2010, «MoDisco : a generic and extensible framework for model driven reverse engineering», dans *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ACM, p. 173–174. 55

[20] Bustard, D., M. Norris et P. Kawalek, éd.. 2000, *Systems Modeling for Business Process Improvement*, 1re éd., Artech House, Inc., Norwood, MA, USA, ISBN 1-58053-050-8. 110

[21] Chardigny, S. 2009, *Extraction d'une architecture logicielle à base de composants depuis un système orienté objet. Une aproche par exploration*, Thèse de doctorat, Université de Nantes. 48

[22] Chein, M. et M.-L. Mugnier. 2009, *Graph-based knowledge representation : computational foundations of conceptual graphs*, Advanced information and knowledge processing, Springer, New York ; London, ISBN 978-1-84800-285-2 978-1-84800-286-9. OCLC : ocn277067628. 37

[23] Cmelik, R. F. et N. H. Gehani. 1988, «Dimensional analysis with c++», *Software, IEEE*, vol. 5, no 3, doi :10.1109/52.2021, p. 21–27, ISSN 0740-7459. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=2021. 29, 70

[24] Consortium, E. 2010, *Java Emitter Templates (JET)*. 64

[25] Cori, R. et D. Lascar. 1993, «Logique mathématique. 1, Calcul propositionnel, algèbres de Boole, calcul des prédicats : cours et exercices», . 34, 37, 41

[26] Coutaz, J. 1997, «PAC-ing the architecture of your user interface», dans *DSV-IS*, p. 13–27. 50, 118

[27] Covington, M. A. 1994, *Natural language processing for Prolog programmers*, Prentice hall Englewood Cliffs (NJ). 56

[28] Cox, M. G., P. M. Harris et I. M. Smith. 2010, *Software specifications for uncertainty evaluation*, National Physical Laboratory. URL http://resource.npl.co.uk/docs/science_technology/scientific_computing/ssfm/documents/dem-es_010.pdf. 26

[29] Cunningham, R. J. et S. Zappacosta-Amboldi. 1983, «Software tools for first-order logic», *Software : Practice and Experience*, vol. 13, no 11, doi :10.1002/spe.4380131106, p. 1019–1025, ISSN 00380644, 1097024X. URL http://doi.wiley.com/10.1002/spe.4380131106. 41

[30] Czarnecki, A. et T. Sitek. 2013, «Ontologies vs. Rules—Comparison of Methods of Knowledge Representation Based on the Example of IT

Services Management», *Information Systems Architecture and Technology : Intelligent Information Systems, Knowledge Discovery, Big Data and High Performance Computing*, p. 99–109. 45

[31] Czarnecki, K. et S. Helsen. 2006, «Feature-based survey of model transformation approaches», *IBM Systems Journal*, vol. 45, nº 3, p. 621–645. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5386627. 62

[32] Davis, R., H. Shrobe et P. Szolovits. 1993, «What is a knowledge representation ?», *AI magazine*, vol. 14, nº 1, p. 17. 6, 34, 35

[33] De Wolf, T. et T. Holvoet. 2004, «Emergence versus self-organisation : Different concepts but promising when combined», dans *International Workshop on Engineering Self-Organising Applications*, Springer, p. 1–15. 50

[34] Diaz, D. et P. Codognet. 1993, «A Minimal Extension of the WAM for clp (FD).», dans *ICLP*, p. 774–790. 102

[35] Eardley, A. et L. Uden, éd.. 2011, *Innovative knowledge management : concepts for organizational creativity and collaborative design*, Information Science Reference, Hershey, PA, ISBN 9781605667010 9781605667027. 36

[36] Eckel, B. 2003, *Thinking in JAVA*, Prentice Hall Professional. 34, 46

[37] EDF, EADS et PhiMeca. 2013, «Reference guide OpenTURNS version 1.1», URL http://trac.openturns.org/wiki/Documentation. 26

[38] Ellison, S. L. R., M. Rosslein, A. Williams et collab.. 2000, «Quantifying uncertainty in analytical measurement», dans *Eurachem/CITAC Guide*, Eurachem. 26

[39] Farrance, I. et R. Frenkel. 2012, «Uncertainty of measurement : A review of the rules for calculating uncertainty components through functional relationships», *The Clinical Biochemist Reviews*, vol. 33, nº 2, p. 49–75, ISSN 0159-8090. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3387884/. 26

[40] Finkelstein, L. 2009, «Widely-defined measurement - an analysis of challenges», *Measurement*, vol. 42, nº 9, doi :http://dx.doi.org/10.1016/j.measurement.2009.03.009, p. 1270 – 1277, ISSN 0263-2241. URL http://www.sciencedirect.com/science/

article/pii/S0263224109000645, <ce :title>Concerning Foundational Concepts of Measurement Special Issue Section</ce :title>. 25

[41] Fitting, M. 2012, *First-order logic and automated theorem proving*, Springer Science & Business Media. 41

[42] Foster, M. P. 2013, «Quantities, units and computing», *Computer Standards & Interfaces*, vol. 35, nº 5, doi :http://dx.doi.org/10.1016/j.csi.2013.02.001, p. 529 – 535, ISSN 0920-5489. URL http://www.sciencedirect.com/science/article/pii/S0920548913000160. 7, 27, 70

[43] Foucart, S. 2006, «Modèle de composants pour le développement d'agents logiciels. MALICE», cahier de recherche, Université de Montpellier II, Université de Montpellier II. 48, 51

[44] Fowler, M. 1997, *Analysis patterns: reusable object models*, Addison Wesley Professional, ISBN 0201895420 9780201895421. 47

[45] Fowler, M. 2010, *Domain-specific languages*, Pearson Education. 7, 54, 55

[46] Frysinger, J. R. «Metric Methods : SI Charts, Visualizing The International System of Units (SI)», URL http://www.metricmethods.com/Resources.php. 98

[47] García-Magariño, I. et G. Palacios-Navarro. 2016, «A model-driven approach for constructing ambient assisted-living multi-agent systems customized for Parkinson patients», *Journal of Systems and Software*, vol. 111, doi :10.1016/j.jss.2015.09.014, p. 34–48, ISSN 01641212. URL http://linkinghub.elsevier.com/retrieve/pii/S0164121215002083. 62

[48] Garlan, D. et M. Shaw. 1993, «An introduction to software architecture», *Advances in software engineering and knowledge engineering*, vol. 1, nº 3.4. 50

[49] Grosan, C. et A. Abraham. 2011, «Rule-Based Expert Systems», dans *Intelligent Systems : A Modern Approach*, édité par C. Grosan et A. Abraham, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-642-21004-4, p. 149–185. URL http://dx.doi.org/10.1007/978-3-642-21004-4_7. 42

[50] Guarino, N., D. Oberle et S. Staab. 2009, «What Is an Ontology ?», dans *Handbook on Ontologies*, édité par S. Staab et R. Studer, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-92673-3, p. 1–17. URL http://dx.doi.org/10.1007/978-3-540-92673-3_0, dOI : 10.1007/978-3-540-92673-3_0. 28

[51] Habra, N., A. Abran, M. Lopez et A. Sellami. 2008, «A framework for the design and verification of software measurement methods», *Journal of Systems and Software*, vol. 81, nº 5, doi :http://dx.doi.org/10.1016/j.jss.2007.07.038, p. 633 – 648, ISSN 0164-1212. URL http://www.sciencedirect.com/science/article/pii/S0164121207002336, <ce :title>Software Process and Product Measurement</ce :title>. 70

[52] Hall, B. D. 2006, «Component interfaces that support measurement uncertainty», *Computer Standards & Interfaces*, vol. 28, nº 3, doi :http://dx.doi.org/10.1016/j.csi.2005.07.009, p. 306 – 310, ISSN 0920-5489. URL http://www.sciencedirect.com/science/article/pii/S0920548905001066. 26

[53] Hand, D. J. 1996, «Statistics and the theory of measurement», *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, vol. 159, nº 3, p. pp. 445–492, ISSN 09641998. URL http://www.jstor.org/stable/2983326. 24

[54] Hannemann, J. et G. Kiczales. 2002, «Design pattern implementation in Java and AspectJ», dans *ACM Sigplan Notices*, vol. 37, ACM, p. 161–173. 47

[55] Harel, D. et B. Rumpe. 2000, «Modeling Languages : Syntax, Semantics and All That Stuff, Part I : The Basic Stuff», cahier de recherche, Weizmann Science Press of Israel, Jerusalem, Israel, Israel. 52

[56] Harel, D. et B. Rumpe. 2004, «Meaningful modeling : what's the semantics of "semantics" ?», *Computer*, vol. 37, nº 10, doi :10.1109/MC.2004.172, p. 64–72, ISSN 0018-9162. 7, 53, 56, 126

[57] Henrio, L., F. Kammüller et M. Rivera. 2009, «An asynchronous distributed component model and its semantics», dans *FMCO (Formal Methods for Components and Objects) - 08*, Sophia antiopolis, France. URL http://hal.inria.fr/inria-00435145. 48, 51

[58] Hodgson, R., P. J. Keller, J. Hodges et J. Spivak. 2013, «QUDT - quantities, units, dimensions and types», URL http://www.qudt.org/. 28, 70

[59] Hollingsworth, D. et U. Hampshire. 1993, «Workflow management coalition the workflow reference model», *Workflow Management Coalition*, vol. 68, p. 26. 110

[60] Horn, A. 1951, «On sentences which are true of direct unions of algebras», *The Journal of Symbolic Logic*, vol. 16, nº 01, doi :10.2307/2268661, p. 14–21, ISSN 0022-4812, 1943-5886. URL https://www.cambridge.org/core/product/identifier/S0022481200102385/type/journal_article. 44

[61] Inc, E. A. 2010, *Sparx Systems*. 55

[62] Ishikawa, K. 1986, *Guide to quality control*, Quality Resources. 26

[63] ISO. 2009, «Iso 80000-1 :2009 quantities and units – part 1 : General», *International Standards Organization*. 23

[64] ISO. 2011, «ISO 19156 :2011 geographic information – observations and measurements», *International Standards Organization*. 27, 70

[65] Jackson, P. 1986, *Introduction to expert systems*, Addison-Wesley Pub. Co.,Reading, MA. 35, 36

[66] Jennings, N. R. 2000, «On agent-based software engineering», *Artificial Intelligence*, vol. 117, nº 2, doi :http://dx.doi.org/10.1016/S0004-3702(99)00107-1, p. 277 – 296, ISSN 0004-3702. URL http://www.sciencedirect.com/science/article/pii/S0004370299001071. 51

[67] Klatt, B. 2007, «Xpand : A closer look at the model2text transformation language», *Language*, vol. 10, nº 16, p. 2008. 64

[68] Knuth, D. E. 1964, «Backus normal form vs. backus naur form», *Communications of the ACM*, vol. 7, nº 12, p. 735–736. 56

[69] Kolski, C., P. Forbrig, B. David, P. Girard, C. Tran et H. Ezzedine. 2009, «Agent-based architecture for interactive system design : Current approaches, perspectives and evaluation», dans *Human-Computer Interaction. New Trends, Lecture Notes in Computer Science*, vol. 5610, édité par J. Jacko, Springer Berlin Heidelberg,

ISBN 978-3-642-02573-0, p. 624–633. URL http://dx.doi.org/10.1007/978-3-642-02574-7_70. 50, 51

[70] Krantz, D., D. Luce, P. Suppes et A. Tversky. 1971, *Foundations of Measurement, Vol. I : Additive and Polynomial Representations*, NY Academic Press. 6, 24

[71] Krötzsch, M., F. Simancik et I. Horrocks. 2012, «A Description Logic Primer», *CoRR*, vol. abs/1201.4089. URL http://arxiv.org/abs/1201.4089. 43

[72] Lamo, Y., X. Wang, F. Mantz, ø. Bech, A. Sandven et A. Rutle. 2013, «DPF Workbench: a multi-level language workbench for MDE», *Proceedings of the Estonian Academy of Sciences*, vol. 62, nº 1, doi : 10.3176/proc.2013.1.02, p. 3, ISSN 1736-6046. 59, 61, 111

[73] Liang, V.-C. et C. J. J. Paredis. 2004, «A port ontology for conceptual design of systems», *Journal of Computing and Information Science in Engineering*, vol. 4, nº 3, doi :10.1115/1.1778191, p. 206, ISSN 15309827. URL http://ComputingEngineering.asmedigitalcollection.asme.org/article.aspx?articleid=1400017. 48

[74] Lloyd, J. W. 1994, «Practical Advtanages of Declarative Programming.», dans *GULP-PRODE (1)*, p. 18–30. 35

[75] Lloyd, J. W. 2012, *Foundations of logic programming*, Springer Science & Business Media. 42

[76] Luce, R. D. et L. Narens. 1994, «Fifteen problems concerning the representational theory of measurement», *Patrick suppes : scientific philosopher*, p. 219–249. 25

[77] Mannion, M. 2002, «Using First-Order Logic for Product Line Model Validation», dans *Software Product Lines : Second International Conference, SPLC 2 San Diego, CA, USA, August 19–22, 2002 Proceedings*, édité par G. J. Chastek, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-45652-0, p. 176–187. URL http://dx.doi.org/10.1007/3-540-45652-X_11, dOI : 10.1007/3-540-45652-X_11. 7

[78] Mari, L. 2013, «A quest for the definition of measurement», *Measurement*, vol. 46, nº 8, doi :http://dx.doi.org/10.1016/j.measurement.2013.04.039, p. 2889 – 2895, ISSN 0263-2241. URL http://www.

sciencedirect.com/science/article/pii/S0263224113001504. 25, 105, 152

[79] Mariem Mahfoudh. 2015, «Adaptation d'ontologies avec les grammaires de graphes typés : évolution et fusion», DOI : 10.13140/RG.2.1.4995.6563. 42

[80] Mazanec, M. et O. Macek. 2012, «On General-purpose Textual Modeling Languages», dans *Proceedings of the Dateso 2012 Annual International Workshop on DAtabases, TExts, Specifications and Objects, Zernov, Rovensko pod Troskami, Czech Republic, April 18, 2012*, p. 1–12. URL http://ceur-ws.org/Vol-837/paper10.pdf. 55

[81] Mehta, N. R., N. Medvidovic et S. Phadke. 2000, «Towards a taxonomy of software connectors», dans *Proceedings of the 22nd international conference on Software engineering*, ACM, p. 178–187. 49

[82] Metzger, A. 2005, «A Systematic Look at Model Transformations», dans *Model-Driven Software Development*, édité par S. Beydeda, M. Book et V. Gruhn, Springer-Verlag, Berlin/Heidelberg, ISBN 3-540-25613-X, p. 19–33. URL http://link.springer.com/10.1007/3-540-28554-7_2. 62

[83] Moalla, N., H. Chettaoui, Y. Ouzrout, F. Noël et A. Bouras. 2008, «Model-Driven Architecture to enhance interoperability between product applications», *PLM08 Proceedings, Seoul, Korea*. 63

[84] Moormann, D., P. J. Mosterman et G. Looye. 1999, «Object-oriented computational model building of aircraft flight dynamics and systems», *Aerospace Science and Technology*, vol. 3, nº 3, doi :http://dx.doi.org/10.1016/S1270-9638(99)80036-4, p. 115 – 126, ISSN 1270-9638. URL http://www.sciencedirect.com/science/article/pii/S1270963899800364. 29

[85] Morley, C. 2007, *Processus métiers et systèmes d'information : évaluation, modélisation, mise en oeuvre*, Dunod, Paris, ISBN 9782100515684 2100515683. 110, 111

[86] Naish, L. 1995, «Pruning in Logic Programming», cahier de recherche, University of Melbourne. 45

[87] Nwana, H. S. 1996, «Software agents : an overview», *The Knowledge Engineering Review*, vol. 11, nº 3, doi :10.1017/S026988890000789X, p. 205–244. 50

[88] OIML, BIPM, CEI, IFCC, ILAC, ISO, UICPA et UIPPA. 2008, «Evaluation of measurement data - guide to the expression of uncertainty in measurement - JCGM 100 :2008 (GUM) - NF ENV 13005 :2009», . 26, 150

[89] OIML, BIPM, CEI, IFCC, ILAC, ISO, UICPA et UIPPA. 2012, «International vocabulary of metrology – basic and general concepts and associated terms (VIM). JCGM 200 :2012 [ISO/IEC guide 99]», . 16, 17, 71, 74

[90] O'Leary, D. E. 1998, «Using AI in knowledge management : knowledge bases and ontologies», *IEEE Intelligent Systems and their Applications*, vol. 13, n° 3, doi :10.1109/5254.683180, p. 34–39, ISSN 1094-7167. 35, 42

[91] OMG, M. 2003, «Guide Version 1.0. 1», *Object Management Group*, vol. 62, p. 34. 59

[92] Parr, T. 2012, *The definitive ANTLR 4 reference*, The pragmatic programmers, The Pragmatic Bookshelf, Dallas, Texas, ISBN 9781934356999 1934356999. 56

[93] Picard, G. 2004, *Méthodologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalité émergente*, thèse de doctorat, Université Paul Sabatier Toulouse III. 50

[94] Pokahr, A., L. Braubach et K. Jander. 2010, «Unifying agent and component concepts : Jadex active components», dans *Proceedings of the 8th German conference on Multiagent system technologies*, MATES'10, Springer-Verlag, Berlin, Heidelberg, ISBN 3-642-16177-4, 978-3-642-16177-3, p. 100–112. URL http://dl.acm.org/citation.cfm?id=1887504.1887519. 48, 50, 51

[95] Rabbi, F., Y. Lamo et W. MacCaull. 2014, «A Flexible Metamodelling Approach for Healthcare Systems», dans *2nd European Workshop on Practical Aspects of Health Informatics (PAHI)*, p. 115–128. URL http://ceur-ws.org/Vol-1251/paper11.pdf. 61

[96] Rabbi, F., Y. Lamo, I. Yu et L. Kristensen. 2015, «A Diagrammatic Approach to Model Completion», dans *Proceedings of the 4th Workshop on the Analysis of Model Transformations co-located with the 18th International Conference on Model Driven Engineering Languages and Systems (MODELS 2015), Ottawa, Canada, September 28, 2015.*, p. 56–65. URL http://ceur-ws.org/Vol-1500/paper7.pdf. 63

[97] Rasiowa, H. et R. Sikorski. 1963, *The mathematics of metamathematics*, Panstwowe Wydawnictwo Naukowe Warszawa. 37

[98] Rasse, A. 2005, *Une Approche Orientée Modèles pour la Spécification, la Vérification et l'Implantation des Systèmes Logiciels Critiques*, Electronique, electrotechnique et automatique, MIPS-Laboratoire Modélisation, Intelligence, Processus, Systèmes, Université de Haute-Alsace. 48, 50, 51, 116, 117, 124

[99] Rijgersberg, H. 2013, *Semantic support for quantitative research*, thèse de doctorat, s.n.], S.l. 28

[100] Rijgersberg, H., M. van Assem et J. Top. 2013, «Ontology of units of measure and related concepts», *Semantic Web*, vol. 4, nº 1, doi : 10.3233/SW-2012-0069, p. 3–13. URL http://iospress.metapress.com/index/M57R51P2417602N2.pdf. 70

[101] Ritchie, C. 2008, *Database principles and design*, Cengage Learning, London, ISBN 978-1-84480-540-2. OCLC : 176925731. 35

[102] Robins, J., D. Redmiles et D. Hilbert. 1999, *ArgoUML*. 55

[103] Roques, A. 2012, *PlantUML*. 55

[104] Rose, L. M., R. F. Paige, D. S. Kolovos et F. A. Polack. 2008, «The epsilon generation language», dans *European Conference on Model Driven Architecture-Foundations and Applications*, Springer, p. 1–16. 64

[105] Royer, J.-C. et M. Xu. 2003, «Analysing mailboxes of asynchronous communicating components», dans *On The Move to Meaningful Internet Systems 2003 : CoopIS, DOA, and ODBASE, Lecture Notes in Computer Science*, vol. 2888, édité par R. Meersman, Z. Tari et D. Schmidt, Springer Berlin Heidelberg, ISBN 978-3-540-20498-5, p. 1421–1438. URL http://dx.doi.org/10.1007/978-3-540-39964-3_89. 50

[106] Rumbaugh, J., I. Jacobson et G. Booch. 2004, *Unified Modeling Language Reference Manual, The (2Nd Edition)*, Pearson Higher Education, ISBN 0-321-24562-8. 54

[107] Rumpe, B. et R. France. 2011, «Variability in UML language and semantics», *Software & Systems Modeling*, vol. 10, nº 4, doi : 10.1007/s10270-011-0210-3, ISSN 1619-1366, 1619-1374. 54

[108] Rutle, A., H. Wang et W. MacCaull. 2013, «A Formal Diagrammatic Approach to Compensable Workflow Modelling», dans *Foundations of Health Information Engineering and Systems*, vol. 7789, édité par D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, J. Weber et I. Perseil, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-642-39087-6, 978-3-642-39088-3, p. 194–212. URL http://link.springer.com/10.1007/978-3-642-39088-3_13. 61

[109] SAMUEL, K., L. OBRST, S. STOUTENBERG, K. FOX, P. FRANKLIN, A. JOHNSON, K. LASKEY, D. NICHOLS, S. LOPEZ, J. PETERSON et e. al. 2008, «Translating OWL and semantic web rules into prolog : Moving toward description logic programs», *Theory and Practice of Logic Programming*, vol. 8, nº 3, doi :10.1017/S1471068407003249, p. 301–322. 45

[110] Sandven, A. 2012, *Metamodel based Code Generation in DPF Editor*, informatique, Department of Informatics, University of Bergen, Department of Computing, Mathematics and Physics Bergen University College, Norway. URL http://dl.acm.org/citation.cfm?id=359723. 64, 111, 126, 127

[111] Schadow, G. et C. J. McDonald. 2009, «The unified code for units of measure», *Regenstrief Institute and UCUM Organization : Indianapolis, IN, USA*. 27, 70

[112] Schnabel, M. C. et S. Watanebe. 2013, «Boost c++ libraries boost.units 1.1.0», URL http://www.boost.org/doc/libs/1_55_0/doc/html/boost_units.html. 29

[113] Sears, A. 1993, «Layout appropriateness : A metric for evaluating user interface widget layout», *IEEE Transactions on Software Engineering*, vol. 19, nº 7, p. 707–719. 110

[114] Sendall, S. et W. Kozaczynski. 2003, «Model transformation: the heart and soul of model-driven software development», *IEEE Software*, vol. 20, nº 5, doi :10.1109/MS.2003.1231150, p. 42–45, ISSN 0740-7459. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1231150. 62

[115] Rodrigues da Silva, A. 2015, «Model-driven engineering: A survey supported by the unified conceptual model», *Computer Languages, Systems & Structures*, doi :10.1016/j.cl.2015.06.001, ISSN

14778424. URL http://linkinghub.elsevier.com/retrieve/pii/S1477842415000408. 8, 57

[116] Smith, F. et M. Proietti. 2014, «Ontology-based Representation and Reasoning on Process Models : A Logic Programming Approach», *arXiv preprint arXiv :1410.1776*. URL http://arxiv.org/abs/1410.1776. 45

[117] Steinberg, D., F. Budinsky, E. Merks et M. Paternostro. 2008, *EMF : eclipse modeling framework*, Pearson Education. 59

[118] Stephenson, A. G., L. S. LaPiana, D. R. Mulville, P. J. Rutledge, F. H. Bauer, D. Folta, G. A. Dukeman, R. Sackheim et P. Norvig. 1999, «Mars climate orbiter mishap investigation board phase I report november 10, 1999», URL http://www.cs.cmu.edu/~./15-610/READINGS/required/casestudies/mco-report-1999.pdf. 3, 21

[119] Stevens, S. S. 1946, «On the theory of scales of measurement», *Science*, vol. 103, nº 2684, p. 677–680. 6, 24, 25

[120] Stobo, J. 1989, *Problem solving with Prolog*, Pitman, London, ISBN 978-0-273-02933-5. OCLC : 19266584. 42

[121] Stroustrup, B. 1995, *The C++ programming language*, Pearson Education India. 34

[122] Sujeeth, A. K., K. J. Brown, H. Lee, T. Rompf, H. Chafi, M. Odersky et K. Olukotun. 2014, «Delite : A Compiler Architecture for Performance-Oriented Embedded Domain-Specific Languages», *ACM Trans. Embed. Comput. Syst.*, vol. 13, nº 4s, doi :10.1145/2584665, p. 134 :1–134 :25, ISSN 1539-9087. URL http://doi.acm.org/10.1145/2584665. 150

[123] Torchiano, M., F. Ricca et P. Tonella. 2010, «Empirical comparison of graphical and annotation-based re-documentation approaches», *IET software*, vol. 4, nº 1, p. 15–31. 55

[124] Tran, C. D., H. Ezzedine et C. Kolski. 2013, «EISEval, a generic reconfigurable environment for evaluating agent-based interactive systems», *International Journal of Human-Computer Studies*, vol. 71, nº 6, doi :http://dx.doi.org/10.1016/j.ijhcs.2013.02.001, p. 725 – 761, ISSN 1071-5819. URL http://www.sciencedirect.com/science/article/pii/S1071581913000098. 50

[125] Vestbø, E., L. Kristensen et Y. Lamo. 2014, *Application of Model Transformations and State Spaces for Verification of Diagrammatic Workflow Models*, thèse de doctorat, Department of Informatics, University of Bergen, Department of Computing, Mathematics and Physics Bergen University College, Norway. URL http://dpf.hib.no/wp-content/uploads/Vestbo14.pdf. 61

[126] Walter, T., F. S. Parreiras et S. Staab. 2009, «Ontodsl : An ontology-based framework for domain-specific languages», dans *International Conference on Model Driven Engineering Languages and Systems*, Springer, p. 408–422. 56

[127] Walter, T., F. S. Parreiras et S. Staab. 2014, «An ontology-based framework for domain-specific modeling», *Software & Systems Modeling*, vol. 13, nº 1, doi :10.1007/s10270-012-0249-9, p. 83–108, ISSN 1619-1374. 42, 56

[128] Wang, A. J. A. 2005, *Component-oriented programming*, Wiley, Hoboken, N.J, ISBN 0471644463. 48

[129] Wang, H., A. Rutle et W. MacCaull. 2012, «A formal diagrammatic approach to timed workflow modelling», dans *Theoretical Aspects of Software Engineering (TASE), 2012 Sixth International Symposium on*, IEEE, p. 167–174. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6269641. 61

[130] Warmer, J. B. et A. G. Kleppe. 1998, «The Object Constraint Language : Precise Modeling With Uml (Addison-Wesley Object Technology Series)», . 56, 60

[131] WFMC. 1999, «Terminology and glossary», cahier de recherche WFMC-TC-1011, Issue 3.0, Workflow Management Coalition. URL http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf. 110

[132] Wielemaker, J. 2009, *Logic programming for knowledge-intensive interactive applications*, thèse de doctorat, University of Amsterdam. Http ://dare.uva.nl/en/record/300739. 42

[133] Wielemaker, J., T. Schrijvers, M. Triska et T. Lager. 2012, «SWI-Prolog», *Theory and Practice of Logic Programming*, vol. 12, nº 1-2, p. 67–96, ISSN 1471-0684. 42, 45, 69, 99

[134] Wolfgang, P. 1994, *Design patterns for object-oriented software development*, Reading, Mass. : Addison-Wesley. 47

Annexe A

Liste des contributions scientifiques

Publications scientifiques

Publications avec comité de lecture

— F. Bourgeois, P. Studer, B. Thirion, and J.-M. Perronne, “A Domain Specific Platform for Engineering Well Founded Measurement Applications”, in Proceedings of the 10th International Conference on Software Engineering and Applications, 2015, pp. 309-318.
DOI :10.5220/0005512103090318

— F. Bourgeois, P. Studer, B. Thirion, and J.-M. Perronne, “Datapipe” presented in the European Project Space, a satellite event of the 10th International Conference on Software Engineering and Applications, 2015. DOI :10.5220/0006163700750096

Présentations sans comité de lecture

— F. Bourgeois, P. Studer, B. Thirion, and J.-M. Perronne, “MEASURE, a platform for adaptive qualified mobile measuring system creation”, Journée Doctorale Sciences de l’Université de Haute-Alsace, 2015

— F. Bourgeois and P. Arlaud, “Datapipe : a Configurable Oil & Gas Automated Data Processor”, in The Success of European Projects using New Information and Communication Technologies, SCITEPRESS, 2016

Soumis

— F. Bourgeois, P. Studer, B. Thirion, and J.-M. Perronne, “A Domain Specific Platform for Engineering Well Founded Measurement Applica-

tions”

Interventions

— Président de session : 10th International Conference on Software Engineering and Applications, 2015

Projets industriels utilisant les travaux

— **Mobile Measure** *Actimage GmbH & Hochschule Offenburg*. Financé par le ministère fédéral de l’économie et de la technologie (BMWI), c’est un AIF Projekt dans le cadre du programme ZIM(Zentrales Innovationsprogramm Mittelstand). Les deux buts du projet sont : l’étude d’approches pour l’utilisation d’instruments de mesures connectés avec des application mobiles ; le développement d’un capteur de mesures 3D de haute précision connecté ainsi que son intégration dans une application mobile. C’est la source des financements de ces travaux. Le projet s’est soldé par la création d’un système de vérification des procédures de mesure conçu à l’aide des règles de la métrologie modélisées en logique du premier ordre.

— **Datapipe** *Actimage Consulting Sas & Dalim Software & Ovation Data Services Inc Uk & Root6 Ltd.*. Datapipe est un projet européen FP7 EUREKA Eurostar de la commission européenne. Le but du projet est de proposer une plateforme pour la gestion, l’analyse, la publication et l’affichage de données issues de l’exploration de poches de gaz ou d’huile. Ces données sont hétérogènes et stockées sur des supports multiples. La plateforme emploie un système automatisant la selection de la procédure à appliquer sur les données à l’aide d’un système employant un ensembe de règles modélisés en logique du premier ordre. Ce système de règle est issu des travaux de recherche. C’est les besoins de ce projets qui ont poussé à intégrer le système de vérification au sein d’une plateforme d’automatisée les interactions entre différents systèmes.

— **Alubar** *Actimage GmbH & Fraunhofer (IZM & IPK) & Siemens AG & Universität Bielefeld (CITEC)*. Le projet ALUBAR (Adaptives Lern- und Unterstützungssystem basierend auf Augmented Reality ou système d’apprentissage adaptif et de soutien basé sur la réalité augmentée) est un projet de recherche innovant soutenu par le ministère fédéral de l’éducation et de la recherche allemand. Il regroupe plusieurs laboratoires de recherches et industriels autour de la théma-

tique de l'utilisation des nouvelles technologies (plateformes mobiles, réalité augmentée, intelligence artificielle) pour accompagner les intervenants dans leurs opérations de manutention nécessitant l'utilisation des deux mains dans des environnements permettant des mouvements restreints. Ce projet à donné naissance à la plateforme Actinote 4.0 de Actimage. Cette plateforme permet la modélisation et l'implantation dans des applications mobiles de procédures assistant les intervenants. Cette plateforme utilise les concepts de modélisation et d'implantation de procédures issues des travaux de thèse. Un système limité pour la vérification de procédures de mesure a également été implanté dans la plateforme.

Annexe B

Liste des acronymes

CGS Système d'unité Centimètre, Gramme, Seconde. 20

DPF *Diagrammatic Predicate Framework*. 59, 61, 111, 148

DSL langage dédié, ou *Domain Specific Language*. 7, 54, 55, 111

EMAML Langage pour la description de configuration d'application de mesure, ou *Executable Measurement Application Model Language*. 7, 11, 121, 149

EMF *Eclipse Modelling Framework*. 59, 60

FOL logique du premier ordre. 37–39, 41, 43, 44, 72

GPL langage générique, ou *General Purpose Language*. 54

GUM Guide pour l'expression de l'incertitude des mesurages. 26

IDM *Ingénierie Dirigée par les Modèles*, ou Model Driven Engineering. 8, 57, 69, 108, 111, 148, 149

IHM Interface Humain-Machine. 110

ISQ système international de grandeurs. 23

JCGM Joint Committee for Guides in Metrology. 17

KR représentation des connaissances. 34–37, 41–45, 68, 72, 76, 145, 151

M2APP Transformation permettant la génération d'une d'application depuis un modèle de spécification, ou *Model to Application*. 8, 10, 125, 129, 149

M2M modèle vers model, ou *Model to Model*. 63

M2T modèle vers texte, ou *Model to Text*. 62, 63, 126, 127

M2VQ Transformation permettant l'extraction de requêtes de vérifications depuis un modèle de spécification, ou *Model to Verification Query*. 8, 125, 149

MAML Langage de modélisation d'application d'assistance à la mesure, ou *Measure Application Modelling Language*. 7, 11, 72, 108–111, 114, 148

MCM modèle conceptuel de la métrologie, ou *Metrology Conceptual Model*. 69, 70, 86

MDA architecture dirigée par les modèles, ou Model Driven Architecture. 58

MKQL Langage de requêtes de vérification de procédures de mesures, ou *Measurement Knowledge Query Language*. 7, 11, 68, 86, 99, 125, 148

MKS Schéma des connaissances de la métrologie, ou *Metrology Knowledge Schema*. 10, 68, 72, 80, 81, 108, 148

NIST National Institute of Standards and Technology. 23

O&M Observations and Measurements. 27

OM Ontology of units of Measure and related concepts. 28

OMG Object Management Group. 59

QUDT Quantities, Units, Dimensions and data Types. 28

RDFS Resource Description Framework Schema. 43

RTM théorie représentationnelle de la mesure. 24, 70, 82

SI Système International. 17, 23, 70

UCUM Unified Code For Units of Measure. 27, 28

UML Unified Modeling Language. 54

VIM Vocabulaire international de métrologie. 17, 70

Annexe C

Glossaire

conversion opération permettant de changer la référence d'une grandeur en une autre. 19

échelle de mesure ensemble des opérations algorithmiques applicables à une grandeur. 24

étalon réalisation de la définition d'une grandeur donnée utilisée comme référence. 18

fonction de mesure Fonction de manipulation algorithmique de grandeurs. 19

grandeur Propriété d'un phénomène exprimable sous forme d'une valeur et d'une référence. 18

grandeur de base grandeur d'un sous-ensemble choisi par convention dans un système de grandeurs donné de façon à ce qu'aucune grandeur du sous-ensemble ne puisse être exprimée en fonction des autres. 23

grandeur dérivée grandeur définie, dans un système de grandeurs, en fonction des grandeurs de base de ce système. 23

incertitude de mesure paramètre non négatif qui caractérise la dispersion des valeurs attribuées à un mesurande. 26

instrument de mesure dispositif utilisé pour faire des mesurages. 2, 19

instrument indicateur dispositif utilisé pour afficher des grandeurs. 19

langage déclaratif langage dont l'ensemble des phrases décrivent le système sans en définir l'exécution. 35

langage impératif langage dont l'ensemble des phrases constituent les actions réalisées par le système développé. 34

mesurage Action de mesurer. 3, 16, 19

métrologie Science des mesurages et ses applications. 6

nature de grandeur Aspect commun à des grandeurs mutuellement comparables. 23

nom spécial nom attribué à une unité dérivée du SI pour en simplifier l'utilisation. 23

procédure de mesure Description détaillée d'un mesurage. 2, 19

référence Représente une quantité connue et fixe d'un évènement observable. Peut être une unité de mesure mais peut également être une procédure de mesure, un matériau de référence, ou une de leurs combinaisons.. 18

résultat de mesure Résultat d'un mesurage. 2

sémantique Signification de l'énoncé. 7, 39

système de grandeurs ensemble de grandeurs associé à un ensemble de relations non contradictoires entre ces grandeurs. 23

système expert système capable de raisonner sur les connaissances d'un domaine et d'en déduire des solutions. 36

unité Grandeur scalaire réelle, définie et adoptée par convention. 18

valeur numérique Valeur associée à une référence pour définir une grandeur. 18

vecteur aux dimensions vecteur dont chaque dimension représente une grandeur de base du système d'unités employé. Il est la représentation canonique assimilable à toute unité d'un système.. 24

widget composants graphiques simples et réutilisables. 110

workflow abstraction dédiée à la modélisation de processus. 110, 111